Manual de uso para “validação.class.asp”

Desenvolvedores

Fernando Tajima Akira

Guilherme Mangabeira Gregio

Luiz Estevam Barros

Neste material iremos detalhar as funcionalidades e a aplicabilidade do arquivo “validação.class.asp”, no qual, dentro do mesmo existem 7 (sete) classes (“ErrosDebug”, “Console”, “Conexao”, “Parametros”, “SqlCommand”, “Token”, “ToolKit”). A classe Erro deve ser utilizada com cuidado, devemos tentar utiliza-la somente para as outras classes internas.

O principal objetivo do uso destas classes é garantimos uma segurança maior nas informações e impossibilitar que Crackers não tenham acesso a tais informações, porém, para aumentarmos este nível de segurança teremos que usar as classes descritas anteriormente. As mesmas devem ser utilizadas para exibição de erros e debug, validação de campos, criação e validação de token entre envio de informações e também realizar a conexão, execução e recuperação de comandos e dados do banco de dados. Porém adicionalmente, foram incluídos funções que podem ajudar no desenvolvimento e resolver alguns problemas de exibição de mensagens para debug do sistema. O arquivo estará disponível inicialmente no diretório “/4Data/Includes/validação.class.asp”.

A seguir descreveremos os métodos públicos de cada classe e quando necessário colocaremos exemplos de sua utilização, depois demonstraremos como realizar o uso das classes, utilizando-se de um formulário para envio de informações com a inclusão do token e uma segunda página para o tratamento das informações e envio ao banco de dados.

1. [Informações Adicionais](#id.vvgb1gxtfck3)
2. [Classe Erros](#id.x48b54w3r9h3)
3. [Classe Console](#id.q47pz681wvxu)
4. [Classe Conexao](#id.1v6oahd09doj)
5. [Classe Parametros](#id.str30axr4u8)
   1. [Tipos](#id.sk2xlj3o8mri)
   2. [Métodos](#id.whvt8ugr7h3c)
6. [Classe SqlCommand](#id.gnfohrjknhmg)
7. [Classe Token](#id.xc5ngcbqqxq6)
8. [Classe ToolKit](#id.kboq9tm75wa1)
9. Exemplos
   1. [Inclusão de Parâmetros](#id.5ru7a5kpphue)
   2. [Conexão e SqlCommand](#id.s8y137oaare1)
   3. [Comandos SQL](#id.bzaqqbr1xm94)
   4. [Executando Comandos](#id.htsoadn0emav)
   5. [Console](#id.aae5i8z5iayq)
   6. [Token](#id.sns8f07mmppy)
   7. [Outras Informações](#id.ounur7hx1907)
   8. [Formulário HTML](#id.pyf3ji93x7a)

***Informações Adicionais***

O arquivo “validação.class.asp” contém uma variável global chamada “setDebugUser”, na qual a classe ErrosDebug e Console fazem sua utilização, assim, quando for verificado que existe um usuário diferente de zero “0” configurado para esta variável serão visualizadas mensagens geradas especificamente da classe ErrosDebug e Console.

As classes ErrosDebug, ToolKit e Token tem objetos instanciados na própria página “validacao.class.asp”, ou seja, após realizar o “include” desta página, você já tem acesso aos objetos (objErros, objToolKit, objToken). Deste modo para se realizar a validação de token e utilizar funções gerais incluídas no “objToolkit”. Uma demonstração de uso esta no exemplo no final do manual.

Importante: Ao realizar o “include” da página existe uma verificação se o usuário está logado, portante se você não estiver logado ou houver perda de sessão, você será redirecionado para a página de login.

***Classe Erros***

*Descrição:*

A classe “Erros” é utilizada pelas outras classes internas, porém o arquivo já contém um objeto desta classe instanciado globalmente chamado “objErros”, que pode ser utilizado para disparar mensagens de erros para os usuários que serão exibidas em uma tela de erros genérica. Devemos realizar o uso consciente destes tratamentos de erros.

*Instância:*

Set [object] = New Erros

*Métodos:*

[object].addError(msg)

Método utilizado para adicionar mensagens de erros a classe, recebe a mensagem de erro como parâmetro (msg).

*[object].closing*

Método que finaliza todos os objetos utilizados pela instância, deve sempre ser utilizado após realizar o uso da instância.

[object].isError()

Método que realiza a verificação de erros na instância da classe, se existirem a execução é finalizada e uma tela com os erros será exibida.

[object].verifyDebugUser

Método que verifica se o usuário logado tem permissão a debugar o sistema, se sim será acrescentado a instrução “On Error Resume Next”. Devemos restringir o uso deste método para as classes internas do arquivo “validacao.class.asp”.

[object]. viewDebugUser

Método que verifica se o usuário logado tem permissão a debugar o sistema e se ocorreu erros de execução, caso exista a execução é finalizada e uma tela com os erros será exibida. A diferença deste método com o método “isError” é que este só será visualizado pelo usuário de debug e serão incluídos erros de execução.

***Classe Console***

*Descrição:*

A classe “Console” é utilizada para realizar o debug do sistema, adicionalmente pode também listar os parâmetros de um objeto “Parametros”. Assim como a classe “Erros” as mensagens só serão exibidas para o usuário com permissão de debug (setDebugUser).

*Instância:*

Set [object] = New Console

*Métodos:*

[object].closing()

Método que finaliza todos os objetos utilizados pela instância, deve sempre ser utilizado após realizar o uso da instância.

[object].console()

Método que exibi uma pagina com as mensagens incluídas e a listagem de parâmetros caso tenham sido incluídos.

[object].printh(str)

Incluí mensagens no console, porém só serão visualizadas quando o método console for chamado e exibidas na pagina de console.

[object].print(str)

Inclui mensagens no console, porém as mensagens também serão exibidas no console e na página na qual está sendo invocado o método.

[object].setParameters(parâmetros)

Inclui um objeto de parâmetros que será listado na pagina de mensagens quando o método console for chamado.

***Classe Conexao***

*Descrição:*

A classe “Conexao” é utilizada para criar uma conexão com o banco de dados e também para trabalhar com transações. É extremamente necessário criar uma instância desta classe, pois a classe “SqlCommand” necessita receber uma instância da mesma para poder executar os comandos.

*Instância:*

Set [object] = New Conexao

*Métodos:*

[object].closing()

Método que finaliza todos os objetos utilizados pela instância, deve sempre ser utilizado após realizar o uso da instância.

[object].getConnection()

Método que retorna uma instância de uma conexão disponível para utilização. A classe “SqlCommand” deve receber em sua propriedade “setConnection” o retorno desta função.

[object].setBeginTrans

Método que inicia transação no banco de dados. Deve ser chamado após o método “getConnection()”.

[object]. setCommandTimeOut(segundos)

Método que seta o tempo de espera para executar um comando, por padrão é configurado com 30 segundos. Deve ser chamado antes do método “getConnection()”.

[object]. setCommitTrans

Método que efeitava o comando dentro da transação no banco de dados. Deve ser chamado após o método “getConnection()”.

[object]. setConnectionString(strSql)

Método configura uma string de conexão para o objeto.

[object].setConnectionTimeOut(segundos)

Método recebe a quantidade de segundos que deverá aguardar enquanto o objeto tenta conectar no banco de dados. Deve ser chamado antes do método “getConnection()”.

[object].setRollBackTrans

Método que cancela a transação no banco de dados. Deve ser chamado após o método “getConnection()”.

***Classe Parametros***

*Descrição:*

A classe “Parâmetros” deve ser utilizada para adicionar parâmetros que serão utilizados por toda a página, assim, os parâmetros adicionados serão validados de acordo com as informações passadas em sua adição e também passarão por validações necessárias para segurança do sistema. Esta classe trabalha somente com alguns tipos de dados que seguem abaixo, no qual são necessários para que o objeto de comando SQL possa validar os tipos de dados passados com o banco de dados.

*Tipos:*

2 - adSmallInt

3 - adInteger

4 - adSingle

5 - adDouble

11 - adBoolean

16 - adTinyInt

20 - adBigInt

129 - adChar

135 - adDBTimeStamp - Data (dd/MM/aaaa) + Hora (hh:mm:ss)

200 - adVarChar

135 - adCusTime – Hora (hh:mm:SS)

135 - adCusDate – Data (dd/MM/aaaa)

135 - adCusTimeStamp – Data (hhMM/aaaa) ou Hora (hh:mm:SS)

200 - adHtml

200 - adXml

*Métodos:*

“Post”, “Get”, ”Session”, ”Cookie”, “Request”

*Instância:*

Set [object] = New Parametros

*Métodos:*

[object]. addRequest(campo, tipo, nulo, tamanho, meth, nome\_method)

Método para adicionar parâmetros com busca direta, utilizando os métodos e tipos descritos acima.

|  |  |
| --- | --- |
| Campo | Nome do parâmetro |
| Tipo |  |
| Nulo | true (pode ser nulo) / false (não pode ser nulo) |
| Tamanho | Para tipos numéricos deve ser zero “0”. |
| Metodo |  |
| Nome\_metodo | Nome do campo que deverá ser recuperado o valor |

[object]. addValue(campo, tipo, nulo, tamanho, valor)

Método para adicionar parâmetros com valores fixos, por exemplo: Variáveis

|  |  |
| --- | --- |
| Campo | Nome do parâmetro |
| Tipo |  |
| Nulo | true (pode ser nulo) / false (não pode ser nulo) |
| Tamanho | Para tipos numéricos deve ser zero “0”. |
| Valor | Valor a ser inserido. |

[object].closing

Método que finaliza todos os objetos utilizados pela instância, deve sempre ser utilizado após realizar o uso da instância.

[object]. getItem(campo, indice)

Método para recuperar os valores adicionados nos parâmetros, através do índice 0 até 4, que são respectivamente ( 0 – campo, 1 – tipo, 2 – nulo, 3 – valor, 4 – tamanho)

[object]. getItemValue(campo)

Método que recupera o valor (índice 3) diretamente do campo solicitado.

[object]. getParameters()

Método que retorna um dicionário com todos os parâmetros adicionados, que será utilizado pela classe SqlCommand.

[object].removeAll()

Método que remove todos os parâmetros do objeto.

[object].removeItem(campo)

Método que remove o item passado como parâmetro do objeto.

[object]. setRequest(campo, tipo, nulo, tamanho, meth, nome\_method)

Método que realiza a atualização de um item no objeto. Se o item passado não existir será gerado erro. O método usa os mesmos parâmetros descritos no método “addRequest”.

[object].setValue (campo, tipo, nulo, tamanho, valor)

Método que realiza a atualização de um item no objeto. Se o item passado não existir será gerado erro. O método usa os mesmos parâmetros descritos no método “addValue”.

***Classe SqlCommand***

*Descrição:*

A classe “SqlCommand” é utilizada para executar comandos SQL no banco de dados. Esta classe deve receber um objeto de “Conexão” através do método “setConnection()”.

*Instância:*

Set [object] = New SqlCommand

*Propriedades:*

[object].cursorLocation

2 – Execução no Servidor

3 – Execução no Cliente (Padrão)

*Métodos:*

[object].closing()

Método que finaliza todos os objetos utilizados pela instância, deve sempre ser utilizado após realizar o uso da instância.

[object]. cmdExecute()

Método que executa uma instrução SQL no banco de dados, caso a instrução for um UPDATe, DELETE ou INSERT retorna a quantidade de registros afetados. Se utilizado um comando SELECT retornará -1.

[object].cmdExecuteScalar()

Método que executa uma instrução SELECT no banco de dados e retorna a primeira célula da primeira linha e primeira coluna dos registros retornados.

[object].cmdInsertGetId()

Método especifico para execução de instrução INSERT no qual retorna o ID gerado no banco de dados. Este método não funciona com Stored Procedures.

[object].getParameters()

Método que retorna um dicionário com os parâmetros incluídos no objeto de comando.

[object]. getRecordSet()

Método que retorna um Recordset. O cursor por padrão será executado no cliente (CursorLocation = 3), para alterar o tipo de recordset deverá ser alterado o CursorLocation antes de sua execução.

*Importante.:* Caso a instrução SQL não retorne registros, o SqlCommand retornará um RecordSet com estado “fechado”, devido a esta particularidade do objeto devemos sempre verificar se o RecordSet está com o estado aberto ou fechado. Caso em alguma parte do código esteja sendo utilizado o “recordset.EOF” e não esteja sendo verificado o estado do recordset anteriormente, ocorrerá erro em tempo de execução.

Estados do RecordSet:

0 - Object is closed

2 - Object is connecting

4 - Object is executing

8 - Object is fetching

1 - Object is open

[object].getStringSql()

Método que retorna a string SQL atual no objeto de comando.

[object].getStringSqlFill()

Método que retorna a string SQL atual no objeto de comando com os valores dos parâmetros preenchidos.

[object]. setConnection(p\_conn)

Método que recebe como parâmetro um objeto do tipo “Conexao”, no qual a classe usará para executar os comandos no banco de dados.

[object].setParameters(parâmetros)

Método recebe um objeto do tipo parâmetros, que será utilizado pelo objeto de comando.

[object.setStringSql(strSql)

Método que configura a string SQL que será executada pelo objeto de comando. Os parâmetros passados na string devem estar entre o sinal de “jogo da velha”. Exemplo: #nome#, #idade#,etc..

***Classe Token***

*Descrição:*

A classe “Token” é utilizada para criar o token de formulário e para realizar a validação do token.

*Instância:*

Set [object] = New Token

*Métodos:*

[object].generateToken()

Método que gera um a string com um token. Aconselhamos sempre utilizar o nome do campo que receberá este valor também como “token”.

[object].generateTokenHidden()

Método que gera um campo do tipo “hidden” já com o token embutido em seu valor.

[object].verifyToken(token)

Método que realiza a verificação se o token é válido, deverá sempre ser verificado no início de recebimentos dos valores dos formulários.

***Classe ToolKit***

*Descrição:*

A classe “ToolKit” possui métodos públicos que podem ser utilizados no sistema, lembrando que um instância desta classe já é criado na inclusão do arquivo “validacao.class.asp”, então para utilizar os métodos basta utilizar o este objeto “objToolkit.[metodo]”.

*Instância:*

Set [object] = New ToolKit

*Métodos:*

[object].showAmountRecords(rs)

Método que retorna um contagem de registros do recordset passado por parâmetro em formato HTML. Exemplo: Resultados: 1 - 20 de 157

[object].verifyLogon()

Método que realiza a validação se o usuário está logado ou não no sistema, caso não esteja, o usuário é redirecionado para página de login.

***Inclusão de Parâmetros***

Criamos um instância da classe parâmetros.

Set parans = New Parametros

Incluindo parâmetros que buscam diretamente seus valores. Para que não tenhamos que fazer o Request.Form, Request.QueryString, Session, etc. Veja que no exemplo abaixo usamos o método “addRequest” para buscar os valores do form diretamente, sem ter que fazer o Request.Form.

parans.addRequest "token","adVarChar",false,100,"post","token"

parans.addRequest "nome","adVarChar",true,100,"Post","nome"

parans.addRequest "idade","adInteger",true,0,"Post","idade"

parans.addRequest “salario”,”adDecimal”, true,0,”Post”,”salario”

parans.addRequest "email","adVarChar",true,100,"PoSt","email"

parans.addRequest "telefone","adVarChar",true,100,"POST","telefone"

parans.addRequest "dateNas","adCusDate",true,0,"post","dateNas"

Podemos fazer a inclusão de parâmetros com valores fixos pelo método “addValue”. Valores que serão passados por QueryString ou que serão incluídos no banco de dados devem ser incluídos nos parâmetros para que possam ser validados.

Dim varstring1 : varstring1 = “valor a ser incluído no banco de dados”

parans.addValue “string1”,”adVarChar”,100,varstring1

Se for necessário alterar algum parâmetro já incluído podemos utilizar os métodos “setRequest” ou “setValue” que recebem a mesma lista de parâmetros.

Para recuperarmos os valores adicionados nos parâmetros devemos utilizar os métodos “getItem” e “getItemValue”. Devemos utilizar estes métodos para incluir os valores nas páginas, em vez de usar variáveis locais.

Parans.getItem(“nome”,3)

Ou

Parans.getItemValue(“nome”)

***Conexão e SQLCommand***

Criamos uma instância da classe de conexão.

Dim connect

Set connect = New Conexao

Se for necessário alterar a string de conexão do banco de dados, use o método setConnectionString. Se não for configurado o objeto utilizara a conexão configurada como padrão.

connect. setConnectionString(“string de conexão”)

Após configurar é necessário termos um objeto SQL Connection para referenciarmos no objeto de SQLCommand, para isso usamos o método “getConnection()”.

Set conn = connect.getConnection()

Criaremos agora um objeto de SqlCommand, para trabalharmos com o banco de dados.

Dim sc

Set sc = new SqlCommand

Faremos a referência do objeto SQLConnection criado anteriormente com a classe SqlCommand, para que o SqlCommand saiba com qual conexão deve trabalhar.

sc.setConnection conn

Agora também temos que realizar a referência do objeto Parametros para o SqlCommand, que utilizara estes parâmetros para executar os comandos no banco de dados.

sc.setParameters parans.getParameters()

Vamos configurar um string SQL que deverá ser executada pelo objeto de comando.

sc.setStringSql "SELECT \* FROM tb\_usuarios WHERE id = 3"

Com o objeto de conexão e sqlCommand configurado, agora podemos finalizar executando o commando.

sc.cmdExecute()

***Comandos SQL***

Os comandos SQL executados pelo objeto de comando tem seus parâmetros configurados de forma diferenciada, isto é necessário para que possamos manter o controle nos dados passados para o banco de dados. Os parâmetros devem ser passados entre o símbolo “jogo da velha”, exemplo: #nome#, #senha#, #usuario#, etc. Abaixo colocaremos alguns exemplos de uso:

SELECT

[object].setStringSql "SELECT \* FROM tb\_usuarios WHERE nome = #nome#"

SELECT LIKE

[object].setStringSql "SELECT \* FROM tb\_usuarios WHERE nome like '%'+#nome#+'%'"

INSERT

[object].setStringSql “INSERT INTO table (campo1) VALUES (#parametro\_campo1#)

UPDATE

[object].setStringSql “UPDATE table SET campo1 = #parametro\_campo1#”

STORED PROCEDURE

[object].setStringSQl “proc\_exec\_CadastroUsuario @cdUsuario = #cd\_usuario#, @nome = #nome#”

***Executando Comandos***

[object].cmdExecute()

Dim sc

Set sc = new SqlCommand

sc.setConnection conn

sc.setParameters parans.getParameters()

sc.setStringSql "INSERT INTO tb\_usuarios(nome) VALUES (#nome#)"

sc.cmdExecute()

[object].cmdExcecuteScalar()

Dim sc

Set sc = new SqlCommand

sc.setConnection conn

sc.setParameters parans.getParameters()

sc.setStringSql "SELECT campo FROM table”

Dim retorno : retorno = sc.cmdExecuteScalar()

[object].cmdInsertGetId()

Dim sc

Set sc = new SqlCommand

sc.setConnection conn

sc.setParameters parans.getParameters()

sc.setStringSql "INSERT INTO tb\_usuarios(nome) VALUES (#nome#)"

Dim id\_adicionado : id\_adicionado = sc.cmdInsertGetId()

[object].getRecordSet()

Dim sc

Set sc = new SqlCommand

sc.setConnection conn

sc.setParameters parans.getParameters()

sc.setStringSql " SELECT nome FROM tb\_usuarios”

Set novoRs = sc.getRecordSet()

While not newRs.EOF

Response.Write("Nome: " & novoRs("nome") & "<br />")

novoRs.movenext

Wend

newRs.Close

***Console***

Devemos utilizar o console para realizar o debug na aplicação, e para centralizarmos as mensagens em um único objeto e em um determinado usuário, deste modo as mensagens poderão ser melhor gerenciadas. Para isso devemos no início da página criarmos umas instância da classe “Console”.

Set c = new Console

Após criado o objeto podemos utilizar os métodos “printh” e “print” para armazenarmos mensagens no console.

c.print("teste")

c.printh("teste hidden")

c.print("testando novalinha")

Com isso podemos chamar o método console para mostrar as mensagens na tela.

c.console()

Obs.: O console só irá funcionar se a variável global “setDebugUser” estiver configurada com um usuário diferente de 0. As informações serão visualizadas para o usuário configurado.

***Token***

Para trocamos informações entre formulários devemos incluir um token e realizar a verificação do mesmo na página que recebera os valores. Para criarmos o token dispomos de 2 métodos, “generateToken” e ‘generateTokenHidden”. A diferença entre eles é que o segundo retorna um campo hidden de formulário com o token já gerado.

generateToken()

<%

Set tk = new Token

%>

<form action="validacao.asp" method="post">

<input type=”hidden” name=”token” value=”<%=tk.generateToken()%>” />

</form>

generateTokenHidden()

<%

Set tk = new Token

%>

<form action="validacao.asp" method="post">

<%=tk.generateTokenHidden()%>

</form>

Quando as informações do Form ou QueryString forem enviados, devemos primeiro realizar a verificação se o token é válido.

<%

Set tk = New Token

tk.verifyToken(parans.getItemValue("token"))

%>

***Outras Informações***

Para trabalharmos com transações no banco de dados, devemos utilizar os métodos setBeginTrans, setCommitTrans e setRollBackTrans da classe de conexão.

[object].setBeginTrans()

...

...instruções

...

IF erro Then [object].setRollBackTrans()

[object].setCommitTrans()

Para verificar se os comandos SQL e os valores passados para o objeto de comando estão corretos, use os métodos “getStringSql” e “getStringSqlFill” da classe SqlCommand. Dica: Você pode utilizar a classe Console para armazenar estas informações.

[object].getStringfSql()

Ou

[object].getStringSqlFill()

Se for necessário colocar alguma instrução no código que só o usuário debug possa ter acesso, podemos utilizar o método “verifyDegubUser()” do objeto globo erros (“objErros”) e adicionalmente enviar as informações utilizando a classe console, assim estaremos garantindo que as instruções só serão executadas para o usuário debug e também que as informações geradas serão visualizadas no console. Exemplo:

<%

Set c = New Console

If ( objErros.verifyDebugUser() ) Then

c.printh(var1 + var2 + var3)

End If

%>

***Formulário HTML***

Segue um exemplo de formulário que inclui o token e realizará o envio das informações para outra página que fará o tratamento.

|  |
| --- |
| <!-- Realizo o include de “validacao.class.asp” que já irá verificar se o usuário está logado -->  <!--#include file="validacao.class.asp" -->  <html xmlns="http://www.w3.org/1999/xhtml">  <head>  <title>SQL Injection</title>  </head>  <body>  <form action="validacao.asp" method="post">  <%=objToken.generateTokenHidden()%>  <input type="text" name="nome" value="guilherme"/><br/>  <input type="text" name="idade" value="24" /><br />  <input type="text" name="salario" value="8233.23" /><br />  <input type="text" name="email" value="email@dlbca.com.br"/><br/>  <input type="text" name="telefone" value="1158400000"/><br/>  <input type="text" name="dateNas" value="20/12/1986"/><br/>  <input type="submit" value="POST"/>  </form>  </body>  </html> |

Na página que será recebida estas informações, temos que realizar o tratamento das informações e incluir no banco de dados, abaixo segue o código comentado da página que fará o tratamento do formulário acima.

|  |
| --- |
| <!-- Realizo o include de “validacao.class.asp” que já irá verificar se o usuário está logado -->  <!--#include file="validacao.class.asp" -->  <%  ' Configuro o usuário que poderá visualizar mensagens de erros e debug  setDebugUser = 12    ' Crio objeto de console  Set c = new Console    ' Crio o objeto para adicionarmos os parâmetros que serão utilizados  Set parans = New Parametros    ' Incluo os parâmetros  parans.addRequest "token","adVarChar",false,100,"post","token"  parans.addRequest "nome","adVarChar",true,100,"Post","nome"  parans.addRequest "idade","adInteger",true,0,"Post","idade"  parans.addRequest "salario","adDouble",true,0,"Post","salario"  parans.addRequest "email","adVarChar",true,100,"PoSt","email"  parans.addRequest "telefone","adVarChar",true,100,"POST","telefone"  parans.addRequest "dateNas","adCusDate",true,0,"post","dateNas"    ' Realizo a validação do token  objToken.verifyToken(parans.getItemValue("token"))    ' Criamos o objeto de conexão  Dim connect  Set connect = New Conexao    ' Criamos uma variavel para recuperar o SQLConnection do objeto de conexão  Set conn = connect.getConnection()    ' Criamos o objeto SqlCommand  Dim sc  Set sc = new SqlCommand    ' Referenciamos o SqlConnection para o objeto SqlCommand  sc.setConnection conn    ' Referenciamos os Parametros para o objeto SqlCommand  sc.setParameters parans.getParameters()    ' Configuramos o comando de inserção que o objeto SQlCommand devera executar  sc.setStringSql "INSERT INTO tb\_usuarios(nome,idade,salario,email,telefone,datenas) VALUES (#nome#,#idade#,#salario#,#email#,#telefone#,#dateNas#)"    ' Crio uma mensagem que será exibida se o console for chamado  c.printh(sc.getStringSqlFill())    ' Com o comando cmdInsertGetId() realizo a inserção no banco de dados e recupero o ultimo id incluído.  Dim novo\_id : novo\_id = sc.cmdInsertGetId()    ' Utilizo o console para armazenar o novo ID gerado, porém como estamos utilizando o método "print", além de armazenar no console a mensagem será exibida também para o usuário configurado no debug  c.print("CODIGO GERADO PELO BANCO: " & novo\_id)    ' Mostro a mensagem que os dados foram inseridos  Response.Write("CD\_INSERIDO: " & novo\_id)    ' Adiciono parametro com o novo ID para realizar a busca  parans.addValue "novoID","adInteger",false,novo\_id,0    ' Como um novo parametro foi incluido devo realizar a referencia dos parametros novamente.  sc.setParameters parans.getParameters()    ' Altero minha string SQL para realizar a busca do dado cadastrado  sc.setStringSql "SELECT \* FROM tb\_usuarios WHERE id = #novoID#"    ' Crio novo recordset e carrego os dados do select  Set novoRecordset = sc.getRecordSet()    ' Verifico se o recordset retornou dados, ou seja, seu estado é diferente de 0  If novoRecordset.state <> 0 Then  ' Realizo o loop no recordset para mostrar os dados  While Not novoRecordset.EOF  Response.Write("<br /><br />Novo Registro<br/>")  Response.Write("ID: " & novoRecordset("id") & "<br />"&\_  "Nome: " & novoRecordset("nome") & "<br />" &\_  "Idade: " & novoRecordset("idade") & "<br />" &\_  "Salario: " & FormatCurrency(novoRecordset("salario")) & "<br />" &\_  "Email: " & novoRecordset("email") & "<br />" & \_  "Telefone: " & novoRecordset("telefone") & "<br />" &\_  "Data Nascimento: " & novoRecordset("dateNas") & "<br />")  novoRecordset.moveNext  Wend  End IF    ' Referencio os parametros para o objeto Console, para que quando for chamado liste também os parâmetros  c.setParameters parans.getParameters()    ' Descomentando o comando abaixo será exebido o console para o usuário debug.  'c.console()    ' Abaixo realiza o fechamento de todos os objetos utilizados, assim liberamos mais recursos para o servidor.  novoRecordset.Close  novoRecordset = Nohting  c.closing  sc.closing  connect.closing  parans.closing  %> |