RTTS, RTTI, RTTC e você: tudo a ver – Parte 3

07/05/2014 09:30

Este é o post que vai agregar valor no seu camarote: você aprenderá como criar *"qualquer parada"* em tempo de execução com o RTTS. Começamos fazendo criações simples, evoluimos para estruturas e terminamos com a criação de tabelas internas.

Para quem chegou de paraquedas na sequência de posts: na Parte 1 você aprende o que esse monte de sigla quer dizer e na Parte 2 aprende a identificar os tipos das variáveis em tempo de execução. Recomendo a leitura desses dois posts antes de fuçar na criação, pois desta vez não vou entrar em detalhes básicos.

Pronto para virar um mestre do RTTS?

Criação de variáveis simples

Para inspecionar uma variável, buscamos a referência no método estático das classes do RTTS utilizando os métodos que começam com "describe_*". Para criar uma variável pelo RTTS, podemos utilizar os **métodos "getters".** Criamos uma variável referenciando o objeto que recebemos do método *"getter"*, e utilizamos field-symbols para acessar seu conteúdo.

Imagine que queremos criar uma variável CHAR de exatamente 18 posições, nem mais, nem menos (nem STRING, para dar "aquele" gato e não se preocupar com tamanho). Utilizando a classe CL_ABAP_ELEMDESCR, conseguimos criar qualquer variável com tipo elementar (integer, string, char, numchar, etc..), do tamanho que quisermos.

Não se assuste com esse papo técnico: analise e entenda o exemplo abaixo. Preste bastante atenção no comando **CREATE DATA**, pois ele tem uma extensão nova específica do RTTS:

```
DATA: elemdescr TYPE REF TO cl_abap_elemdescr.

DATA: variavel  TYPE REF TO data.

FIELD-SYMBOLS <char> TYPE any.

* Criando uma variável CHAR de 18 posições.
elemdescr = cl_abap_elemdescr=>get_c( 18 ).

* Aqui a magia do TYPE HANDLE entra em ação. Ele foi
* feito especialmente para funcionar com as classes do RTTS
* O comando abaixo quer dizer:
* CRIE uma variavel DO TIPO dessa classe rtts.
CREATE DATA variavel TYPE HANDLE elemdescr.

* Você não vai conseguir fazer mta coisa com a variável type ref to data.
* Precisamos utilizar um fieldsymbol para manipular o valor.
ASSIGN variavel->* TO <char>.

* Sao 18 posicoes, então o 9 teria que ser cortado.
* Teste e veja se o output está correto!
<char> = '1234567890123456789'.
WRITE <char>.
```

➡ **Conceito importante:** somente instanciar a classe RTTS não cria, de fato, a variável. Quem aloca a memória e cria variável é o comando **CREATE DATA... TYPE HANDLE .** O escopo da variável (global/local) segue a declaração do "<nome> TYPE REF TO data". Se você declarar isso como global, a variável será global, se declarar como local, ela será local. Entenda isto e evite confusões!

Mas e aí, o código acima pareceu muito trabalhoso?

Para o exemplo de uma só variável, fazer todo esse rolê maluco para criar um CHAR de 18 parece insano. Mas não subestime a flexibilidade do RTTS! Modificando um pouco o exemplo acima, **podemos criar uma máquina de gerar variáveis**. Veja só:

```
DATA: elemdescr TYPE REF TO cl_abap_elemdescr.

* Char
elemdescr = cl_abap_elemdescr=>get_c( 10 ).
PERFORM escreva_sei_la_o_que USING elemdescr '1234567890A'.

* Inteiro
elemdescr = cl_abap_elemdescr=>get_i( ).
PERFORM escreva_sei_la_o_que USING elemdescr 123.

* NumChar
elemdescr = cl_abap_elemdescr=>get_n( 5 ).
PERFORM escreva_sei_la_o_que USING elemdescr 987654.

* String
elemdescr = cl_abap_elemdescr=>get_string( ).
PERFORM escreva_sei_la_o_que USING elemdescr 'Prevenindo Consultores e Virarem Zumbis'.

* Packed com Decimais
elemdescr = cl_abap_elemdescr=>get_p( p_length = 10 p_decimals = 5 ).
PERFORM escreva_sei_la_o_que USING elemdescr '1234567890.12345' .

*&---------------------------------------------------------------------*
*&      Form  ESCREVA_SEI_LA_O_QUE
*&---------------------------------------------------------------------*
FORM escreva_sei_la_o_que  USING p_elemdescr TYPE REF TO cl_abap_elemdescr
                                 p_seila.

  DATA: variavel TYPE REF TO data.

  FIELD-SYMBOLS <variavel> TYPE any.

  CREATE DATA variavel TYPE HANDLE p_elemdescr.

  ASSIGN variavel->* TO <variavel>.

  <variavel> = p_seila.
  WRITE: /01 'Tipo:',
          7   p_elemdescr->type_kind,
          20 'Tamanho:',
          35  p_elemdescr->length,
          50 'Valor:',
          60 <variavel>.

ENDFORM.                    " ESCREVA_SEI_LA_O_QUE
```

Viu só? Eu fiz um mini-form para reutilizar o código com o CREATE DATA, ASSIGN e o WRITE. Isso **aumentou a flexibilidade** do meu código, pois o meu form não faz a menor idéia de que tipo de variável ele vai escrever. Mas isso não importa, afinal, qualquer variável criada pela classe CL_ABAP_ELEMDESR pode ser escrita na tela por ele.

Notem que no exemplo eu **não estou guardando as variáveis criadas** em lugar nenhum, só estou gerando uma variável, escrevendo o seu valor e esquecendo que ela existe. Na sua implementação você pode guardar os valores declarando variáveis "TYPE REF TO data" **diferentes** para cada uma das variáveis que você quiser guardar, com o escopo que preferir (global/local).

Outro ponto importante, é notar que a classe CL_ABAP_ELEMDESCR possui um método "*getter*" específico para cada tipo elementar. Se eu quero um integer, utilizo o **GET_I,** se quero uma string, o **GET_STRING**, e **assim por diante**. Mas como fica quando eu quiser criar algo com base no DDIC, onde uma hora a variável vai ser char, outra hora int, outra hora numchar...?

Criação de variáveis com base no DDIC

Deixa eu te contar uma coisa sobre o método "*describe_by_name*": sabia que ele é o suficiente para você ter uma referência real de uma variável do DDIC? **Ele não serve só para inspeção não!** (os exemplos da parte 2 ficaram muito mais interessantes agora, certo? 🙂)

No caso da criação com base no DDIC, precisamos primeiro **descrever** um elemento de dados numa das classes do RTTS. Por este motivo, utilizamos a class CL_ABAP_DATADESCR e o método "*describe_by_name*" para referenciar um elemento de dados do DDIC. Depois é só fazer o CREATE DATA..TYPE HANDLE e o ASSIGN para utilizar a variável.

Simples assim!

```
DATA: datadescr TYPE REF TO cl_abap_datadescr.
DATA: variavel  TYPE REF TO data.

FIELD-SYMBOLS: <variavel> TYPE any.

* Pelo describe_by_name, conseguimos mostar qual elemento de dados
* o RTTS deve usar de referência para instanciar o objeto.
datadescr ?= cl_abap_datadescr=>describe_by_name( 'MATNR' ).

* Depois utilizamos o mesmo esquema para conseguir passar valor
* para a variável
CREATE DATA variavel TYPE HANDLE datadescr.
ASSIGN variavel->* TO <variavel>.

* Se a exit de conversão estiver ativa, você verá o output
* desse valor sem os zeros, afinal, o campo é um MATNR!
<variavel> = '000000000012345678'.
WRITE <variavel>.
```

Agora, pegue o exemplo da máquina de criar variáveis e aplique no exemplo do DDIC. Será que é possível? Será que dá para fazer outras coisas doidas aliando o RTTS e métodos/form ? 😉

Mas Mauricio, eu só tenho o nome "tabela-campo", como que eu vou fazer para criar uma variável com base no DDIC só com o campo da tabelaaaaaaaa??????

Peraí... você tem a **TABELA** e o **CAMPO**? Então é só descobrir se ele faz referência ao DDIC ou a um tipo pŕe-definido (quando você coloca o tipo direto na SE11), e criar o campo utilizando as classes RTTS que já estudamos. O que? Você quer saber como descobrir essas coisas?

Pô zumbi, usa o google, tá achando que a vida é fácil? 🙄 😈 Zueira, está aqui o link.

Criação de Estruturas

Já vimos anteriormente que as classes do RTTS possuem uma sinergia que atinge mais de 9000. Então, não é surpresa nenhuma descobrir que para criar uma estrutura com RTTS, você primeiro precisa criar os campos indivudalmente, para só então organizá-los em uma estrutura com a classe CL_ABAP_STRUCTDESCR.

No exemplo abaixo criamos uma estrutura com dois campos, um inteiro e um campo do DDIC:

```
DATA: estrutura TYPE REF TO data.

FIELD-SYMBOLS: <estrutura> TYPE any,
               <campo>     TYPE any.

DATA: structdescr TYPE REF TO cl_abap_structdescr.

* O get da CL_ABAP_STRUCTDESCR tem um paramêtro chamado "P_COMPONENTS".
* Não precisa ser nenhum PhD para entender o que precisamos fazer..
DATA: componentes TYPE abap_component_tab,
      componente  LIKE LINE OF componentes.

* A importância de entender o RTTS por "etapas" como estamos fazendo
* vem agora. A estrutura de componentes tem um campo chamado "TYPE", que
* aceita um objeto do tipo CL_ABAP_DATADESCR. Nós já aprendemos como
* criar objetos que fazem referência a variáveis, certo? Então vamos
* montar nossa estrutura!

DATA: datadescr TYPE REF TO cl_abap_datadescr,
      elemdescr TYPE REF TO cl_abap_elemdescr.

* Criando um campo CHAR de 10
elemdescr = cl_abap_elemdescr=>get_c( 10 ).
componente-name = 'CHAR_DE_10'.
componente-type = elemdescr.
APPEND componente TO componentes.

* Criando um campo para a planta, com o elemento de dados WERKS_D
datadescr ?= cl_abap_datadescr=>describe_by_name('WERKS_D').
componente-name = 'WERKS'.
componente-type = datadescr.
APPEND componente TO componentes.

* Agora vamos criar nossa estrutura!
structdescr = cl_abap_structdescr=>get( componentes ).

* Mesmo esquema de sempre para poder acessá-la
CREATE DATA estrutura TYPE HANDLE structdescr.
ASSIGN estrutura->* TO <estrutura>.

* Porem para acessar os campos e preenche-los, vamos utilizar o
* ASSIGN COMPONENT utilizando um índice para indicar qual o campo
* que queremos acessar
ASSIGN COMPONENT 1 OF STRUCTURE <estrutura> TO <campo>.
<campo> = '1234567890'.

ASSIGN COMPONENT 2 OF STRUCTURE <estrutura> TO <campo>.
<campo> = '3000'.

* Pare aqui no debug e veja os valores preenchidos em <estrutura> :)
BREAK-POINT.
```

E está pronta a nossa estrutura criada dinamicamente. Se você não conhecia o ASSIGN COMPONENT, leia este post aqui do blog onde dissecamos o comando ASSIGN.

Vamos em frente: primeiro você cria alguns campos, depois cria a estrutura utilizando os campos, e depois...

Criação de Tabelas Internas

Agora é só pegar a estrutura do exemplo acima e utilizá-la para criar uma tabela interna! Este exemplo serve também como uma **sumarização de tudo que aprendemos** sobre criação de variáveis neste post.

Se você só sabia criar tabela interna com aquele método do ALV, saiba que é muito mais legal quando a gente entende o que acontece lá dentro.

Este exemplo também cobre o preenchimento da tabela criada dinâmicamente, afinal, para que serve uma tabela interna se eu não souber como preenchê-la? 🙂

Divirta-se!

```
DATA: tabela    TYPE REF TO data,
      estrutura TYPE REF TO data.

FIELD-SYMBOLS: <tabela>    TYPE standard table,
               <estrutura> TYPE any,
               <campo>     TYPE any.

DATA: structdescr TYPE REF TO cl_abap_structdescr,
      tabledescr  TYPE REF TO cl_abap_tabledescr,
      datadescr TYPE REF TO cl_abap_datadescr,
      elemdescr TYPE REF TO cl_abap_elemdescr.

DATA: componentes TYPE abap_component_tab,
      componente  LIKE LINE OF componentes.

*-- Criação da Estrutura Igual Exemplo Anterior:

elemdescr = cl_abap_elemdescr=>get_c( 10 ).
componente-name = 'CHAR_DE_10'.
componente-type = elemdescr.
APPEND componente TO componentes.

datadescr ?= cl_abap_datadescr=>describe_by_name('WERKS_D').
componente-name = 'WERKS'.
componente-type = datadescr.
APPEND componente TO componentes.

* Agora vamos criar nossa estrutura!
structdescr = cl_abap_structdescr=>get( componentes ).

* Mesmo esquema de sempre para poder acessá-la
CREATE DATA estrutura TYPE HANDLE structdescr.
ASSIGN estrutura->* TO <estrutura>.


*-- Criação da Tabela

* Ok, vamos criar essa tabela bunita. É claro que vamos utilizar o GET da
* clase CL_ABAP_TABLEDESCR. Notou que tudo no RTTS segue o mesmo padrão
* de funcionamento?

* O parâmetro obrigatório se chama P_LINE_TYPE. Podemos passar nossa estrutura!
tabledescr ?= cl_abap_tabledescr=>get( structdescr ).
```

```
CREATE DATA tabela TYPE HANDLE tabledescr.
ASSIGN tabela->* TO <tabela>.

* Você ainda pode dizer qual o tipo da tabela (SORTED, HASHED, STANDARD) e quais
* os campos chaves da sua tabela interna. Experimente fuçar no método "get".



*-- Preenchendo a tabela criada dinâmicamente

* Neste caso, eu coloquei o field-symbol como TYPE STANDARD TABLE. Isso me permite
* dar um APPEND da estrutura  pois eu criei a tabela interna da
*  forma default (que é STANDARD TABLE).

* Primeiro preechemos a estrutura:
ASSIGN COMPONENT 1 OF STRUCTURE <estrutura> TO <campo>.
<campo> = 'MATERIAL123'.

ASSIGN COMPONENT 2 OF STRUCTURE <estrutura> TO <campo>.
<campo> = 'SP01'.

* Depois é só dar o append!
APPEND <estrutura> TO <tabela>.

* Pare aqui no debug e veja a tabela criada de forma totalmente dinâmica, já
* preenchida com uma linha! :)
BREAK-POINT.
```

Você ainda pode utilizar a tabela interna criada para jogar os dados de um SELECT. É uma tabela interna e você pode fazer o que quiser com ela!

Muita gente procura na internet como criar tabelas dinâmicas, caem nos exemplos do RTTS e ficam sem entender absolutamente nada. Mas falaí: quando você entende a sequência de criação e a lógica da hierarquia de classes, fica bem mais fácil.

E aí, tem mais?

Sim, tem muito mais. Minha intenção é que esta sequência de posts consiga lhe mostrar a importância de compreender o RTTS para ser capaz de criar aplicações que fazem coisas malucas. (Ou, pelo menos, deixar *aquele* dev com cara de WTF quando olhar o seu programa 😀).

O RTTS pode criar outras coisas além destas explicadas (como instâncias de classes!). Pesquise na hierarquia RTTS, que descrevemos na primeira parte.

Como sugestão de estudo para aprimorar os conhecimentos, tente aliar o que aprendeu na parte 2 com a parte 3 para criar uma estrutura a partir de um TYPES declarado no seu programa.

Eu também cobri a criação utilizando os *getters* e com base no *describe_by_name,* mas você irá notar que as classes RTTS possuem métodos com o prefixo *create.* Descubra lendo o help das classes porque eu ignorei a criação com os *creates* .

Se você tiver alguma experiência legal com RTTS e quiser compartilhar, comente! Aproveite este espaço deixar registrado a sua sugestão de uso para os seus camaradas zumbis.

Baixe todos os exemplos

Acesse a página do ABAPZombie no Github para baixar todos os exemplos desta sequencia de posts.

Sempre que você precisar de um código aqui do site, acesse o link “Códigos ABAPZombie” no menu lateral. Fique a vontade para nos enviar correções direto no repositório do Github!

Acaboooooou!

Foram 3 semanas, muito código, muitas siglas e muitas piadas idiotas... mas chegamos ao fim! Se você gostou, compartilhe via facebook, twitter, google plus, telefone, sms, sinal de fumaça, grite para o amigo do lado, pombo correio – qualquer jeito ajuda.

Abraços a todos aqueles que gostaram de sequência mas não entenderam a piada do título 😕

Comentários

Daniel Jesus — *28/05/2014 09:19*

RTTS é mara!
Dá um trabalhinho pra pensar e organizar no inicio (assim como qq coisa com OO), mas depois fica supimpa.
Precisei uma vez criar relatorios dinamicos para calculo de depreciação....o funcional até hoje me ama ! (coisa rara no mundo ABAP)

Henrique Dias — *07/05/2014 18:02*

Muito boa a série.

Por acaso eu fiz uma classe louca que da um submit num programa que gera um alv que busca as informações desse alv com uma classe maluca que retorna uma tabela dinamica que tem que ser identificada usando essas classes doidonas para transformar isso em alguma coisa que a classe inicial pudesse usar.

Quem sabe um dia eu transformo isso num exemplo bacana pra postar.

Abraços a todos que não entenderam nada na frase sem pontuação

> **Mauricio Cruz** — *07/05/2014 19:12*
>
> Valeu manolo que bom que voce gostou espero que voce possa ajustar essa parada ae que voce criou para compartilhar com os seus amiguinhos eu acho que entendi o que voce fez mas talvez nao tenha entendido danem se os pontos virgulas e etcs abs