PROGRAMAR

A REVISTA PORTUGUESA DE PROGRAMAÇÃO

Revista nº 20 - JUNHO de 2009 www.portugal-a-programar.org

Metaprogramação em C++

Conheça as verdadeiras potencialidades dos templates de C++

Pág 4

Google Web Toolkit - Parte 2

Continuação do tema de capa da edição anterior

Pág 14

Arduino e a Aquisição de Dados

Utilizações avançadas do Arduino na aquisição de dados

Pág 24

índice

equipa PROGRAMAR

coordenadores
Joel Ramos
Pedro Abreu

editor
António Silva

capa
Maurício Rodrigues

redacção
Cristian Gonçalves
Fábio Ferreira
Fábio Pedrosa
Francisco Almeida
Nuno Santos

equipa de revisão
Bruno Oliveira
Fernando Martins
Miguel Rentes
Nuno João

equipa de divulgação
David Ferreira

contacto
revistaprogramar
@portugal-a-programar.org

website
www.revista-programar.info

issn
1647-0710

Como fazer uma montanha andar?

O Departamento de Ciência de Computadores da Faculdade de Ciências da Universidade do Porto (DCC-FCUP) recebeu, a 29 de Maio, a final das Olimpíadas Nacionais de Informática (ONI). Realizadas em Portugal desde 1989, são a mais representativa competição para alunos do secundário e é nesta final que se inicia tipicamente o processo de selecção da delegação portuguesa para as Olimpíadas Internacionais de Informática (IOI), a realizar em Agosto. Nos últimos anos, seguiu-se às ONI um estágio para os alunos seleccionados, de forma a prepará-los para o nível competitivo que encontram nas IOI e concluir a selecção dos alunos, escolhendo-se os quatro melhores para a prova internacional.

Este ano, a prova foi organizada sob circunstâncias excepcionais. A Caixa Geral de Depósitos, principal patrocinador em edições anteriores, retirou-se. Seguiu-se então a procura de novos apoios, mas apesar do esforço por parte da organização, os patrocinadores encontrados não devem substituir a ajuda financeira com que a Caixa Geral de Depósitos contribuía, o que conduz à situação actual. A redução dos recursos fez já uma grande vítima - não haverá estágio para os seleccionados, pelo menos nos moldes habituais - mas ameaça também a prova internacional, existindo a possibilidade de Portugal não poder apresentar nas IOI a totalidade dos concorrentes seleccionados.

Torna-se, desta vez, ainda mais difícil conseguir os tão ansiados resultados, ainda para mais competindo contra concorrentes com anos de preparação. É certo que os tempos são de crise e a decisão da Caixa Geral de Depósitos não pode ser condenada; sabe-se também que a informática não é um dos mais privilegiados sectores na educação e as ONI ainda são tidas por muitos em menor consideração que as mais nobres Olimpíadas da Matemática e, eventualmente, da Física. É definitivamente condenável, no entanto, que se tenha atingido uma situação destas, ao ponto de estar em risco a participação nas IOI de alguns dos alunos apurados e de não se assegurarem condições minimamente competitivas de preparação para o nível internacional. Alguma empresa irá, no futuro, absorver cada um destes jovens programadores e, de certa forma, beneficiar da sua experiência nesta vertente. Porque não investir então na sua preparação, porque não suportar a escalada dos concorrentes portugueses rumo às tão longínquas mas tão atingíveis medalhas nas IOI?

Em primeiro lugar, há que melhorar a visibilidade dada aos patrocinadores. Ter um grande logótipo no site, nas t-shirts e no local da final não é, na verdade, um grande retorno publicitário para um bom patrocínio, pelo que apoiar o evento não se revela claramente vantajoso para as empresas. O actual palmarés português nas IOI também não melhora a situação, o que nos traz a inevitável relação recursos-resultados. A falta de resultados pode diminuir os apoios, principalmente a nível financeiro, mas a falta de apoios impede uma boa preparação e limita os desempenhos dos concorrentes, acabando por amputar as suas ambições. É perfeitamente previsível que um bom investimento inicial abriria caminho a uma série de bons resultados, e essa série de bons resultados traria definitivamente mais visibilidade aos patrocinadores, principalmente através da comunicação social. O que falta é só esse mesmo investimento. E enquanto não houver alguém capaz de o perceber, enquanto a vontade de obter resultados por parte dos concorrentes e o nível de preparação a que são sujeitos se mantiverem tão distantes, igualmente distantes continuarão os resultados. Não só nas IOI, não só nas competições; tudo isto se aplica na construção de um futuro e de uma sociedade capaz de ambicionar sempre o próximo passo. Por esse futuro esperamos.

Pedro Abreu

Lançamento do Clojure 1.0

http://clojure.blogspot.com/2009/05/clojure-10.html

Course | Programming Methodology

Metodologia de Programação é uma introdução à engenharia de aplicações informáticas com ênfase em princípios modernos de engenharia de software: orientação a objectos, estruturação, encapsulamento, abstracção e testes. Usa-se a linguagem Java, com destaque para um bom estilo de programação e para o uso das funcionalidades presentes na linguagem.

http://www.youtube.com/view_play_list?p=84A56BC7F4A1F852

Google AJAX Libraries API

http://code.google.com/intl/pt-PT/apis/ajaxlibs/

Próximos eventos

Carnegie Mellon Portugal Summer Academy 2009

http://www.cmuportugal.org/summeracademy/

Olimpíadas Nacionais de Informática 2009

• 3 dos 4 finalistas que poderão representar Portunal nas International Olympiad in Informatics (IOI) são membros do P@P, sendo que 1 deles é um dos coordenadores desta revista, e outro é um redactor.

http://www.dcc.fc.up.pt/oni/2009/

Metaprogramação em C++

Introdução

Toda a gente que conhece minimamente a linguagem C++ já ouviu falar em templates. O conceito inovador de template foi oficialmente introduzido no standard de implementação em 1998 e trouxe uma lufada de ar fresco, tanto ao C++, como a um número de outras linguagens mais recentes (por exemplo, também linguagens como o Java ou C# foram enriquecidas com as suas próprias técnicas de programação genérica). Neste artigo, iremos rever os templates, bem como as suas propriedades, e explorar algumas das possibilidades raramente consideradas com templates.

O que são e o que fazem templates

O verdadeiro potencial dos templates é que nem sempre foi evidente. Mas isso mudou com o tempo. Em primeiro lugar, vejamos com o próximo exemplo, uma utilização clássica de templates, com base nas suas capacidades de parametrização.

```
template<class T>
T max(const T &a, const T &b)
{ return a > b ? a : b; }

template<class T>
T sqr(const T &a)
{ return a*a; }

...

int n = 2, m = 5;
max<int>(n,m);    // Devolve 5

double x = 0.67, y = 2.71828;
max(x,y);         // Devolve 2.71828

int r = 12;
sqr(r);           // Devolve 144
```

O que está em cima são dois típicos exemplos de livro teórico de templates de funções e respectivas instanciações. Essencialmente, o que um template faz é permitir escrever uma função de forma parametrizada, isto é, antes de saber de que tipo são as variáveis envolvidas. Isto é bastante útil no sentido em que de uma só vez se garantiu a implementação das funções max e sqr para qualquer tipo de variável (a variável genérica T). É de salientar outra característica importante dos templates: o seu mecanismo de instanciação. Um template de uma função não é uma função, mas um protótipo. Isto é, o compilador verifica a sintaxe, mas ignora a validade lógica de um template. Isto até que o template seja chamado no código. Por exemplo, a função max não existe em parte alguma do código antes de ser instanciada ao chamarmos max<int>. E novamente, ao chamarmos max<double>. Ou seja, o compilador tomou o protótipo da função max e de cada vez que o programador pediu uma nova forma da função, utilizou-o para criar uma nova função max em overload. Como devem ter reparado, podemos chamar max<double> sem especificar o tipo double, porque o compilador é capaz de deduzir que versão do template deve ser usada a partir dos argumentos da função. Apenas exigimos para esta função que ambos os argumentos sejam do mesmo tipo T.

Para além de templates de funções, existem templates de classes. E, similarmente, cumprem a mesma tarefa que os templates de funções: permitem declarar uma classe de forma genérica. A classe genérica Sequencia implementada neste exemplo revela a sintaxe a que poderíamos recorrer para parametrizar uma classe em função do tipo de variável do conteúdo.

```
template<class T = double>
class Sequencia
{
  private:
    long quant;
    T *p;
  public:
    Sequencia(const long n = 10)
    { quant = n; p = new T[n]; }
    virtual ~Sequencia()
    { delete[] p; }
};

...

Sequencia<double> s1;
Sequencia<unsigned long> s2;

Sequencia s3;   // Instanciação
automática a double

Sequencia<Sequencia> s4;  // possível
```

O que este template de classe tem de interessante é o valor automático para o tipo. Isto quer dizer que se omitirmos o parâmetro de tipo na declaração do template, o compilador seguirá a indicacão dada de que por omissão, o conteúdo é do tipo double. Outro pormenor interessante é que até é possível criar um objecto Sequencia de objectos Sequencia. Tal é a flexibilidade dos templates.

Propriedades dos templates

Até agora, não temos visto nada de especial, a não ser uma funcionalidade que até pode vir a dar jeito para certos tipos de funções e estruturas. É agora altura de reflectir sobre as propriedades dos templates:

- Aceitam como parâmetros tipos genéricos (class ou typename), apontadores para funções, ou constantes literais inteiras ou booleanas (ou seja, desde que o valor seja conhecido aquando da compilação).
- Ao serem instanciados, os parâmetros-tipo são verificados contra o código utilizado.
- Podem utilizar qualquer número de parâmetros, e estes podem ser deduzidos.
- Podem ser instanciados recursivamente.

Estas propriedades são muito interessantes. Na prática, um mecanismo de instanciação que permite utilizar valores inteiros, e ainda para mais, recursivamente, é o mesmo que uma linguagem de programação em si. Este resultado abre portas a uma infinidade de possibilidades: o mecanismo de instanciação dos templates é funcionalmente similar a uma máquina de Turing. Por outras palavras, os templates podem levar um compilador de C++ a comportar-se como um interpretador! Isto demonstrar-se-á na próxima secção.

Um primeiro metaprograma

Sabendo agora que um template também aceita inteiros (atenção, constantes literais) como parâmetro, e sabendo da sua recursividade, vamos experimentar um pouco o conceito de metaprogramação fazendo o compilador calcular o factorial de um número.

```
template<int N>
class Factorial
{
  public:
    static inline int valor()
    { return N*Factorial<N-
1>::valor(); }
};

// Isto é uma especialização:
template<>
class Factorial<1>
{
  public:
    static inline int valor()
    { return 1; }
};

...

Factorial<6>::valor();
// Devolve 6! (ou seja, 720)
```

O aspecto interessante neste exemplo é que uma vez compilado o código, o método estático Factorial<6>::valor() não faz qualquer chamada a funções, e limita-se a devolver a constante 720. O compilador instanciou 6 classes Factorial<N>, uma para cada valor de N. Cada classe forçou o compilador a criar a classe de ordem inferior, decrementando o N sucessivamente até à última especialização Factorial<1>. Durante a compilação, os valores N, N-1, ... , 1 são sucessivamente multiplicados dando finalmente origem ao valor 720 como uma constante literal. O factorial é calculado durante a compilação.

Aplicações de templates

Sejamos sinceros, por muito engraçado que seja coagir o compilador a calcular um factorial ou a gerar números primos, é óbvio que fazê-lo no compilador não tem grande utilidade prática. Nesse caso, perguntar-se-ão muitos, para que mais servirão templates, afinal? Como é possível prever no exemplo dado anteriormente, um template de uma classe ou estrutura pode ser tratado como um programa que gera código durante a compilação. Através da especialização, é por exemplo possível criar classes cuja implementação depende da plataforma ou de um parâmetro do sistema, de uma forma mais elegante que utilizando macros: ao invés de se limitarem a fazer substituição de texto, os templates passam argumentos e verificam os tipos.

```
template<typename T, bool Windows_Vista>
class AMinhaClasse
{
  private:
    T _idAlgumaCoisa
  public:
    ...
};

// As especializações verificam a
versão:
template<typename T>
class AMinhaClasse<T,true>
{
  private:
    T _idAlgumaCoisa
  public:
    const bool Especifico()
```

```
    {
      // Código específico para
Vista...
      // ... ou 7, claro ;)
    }
};

template<typename T>
class AMinhaClasse<T,false>
{
  private:
    T _idAlgumaCoisa
  public:
    const bool Especifico()
    {
      // Código genérico para XP...
    }
};

...

// Em qualquer versão de Windows:
const bool is_vista =   (VERSAO_WIN >=
6);
// Isto compila se is_vista é
constante!
AMinhaClasse amc<long,is_vista>;
amc.Especifico();
```

Olhando para este pedaço de código, que é possível obter o mesmo comportamento com macros de #if, podemo-nos perguntar quanto à sua relevância. Ora bem, a verdade é a instanciação de AMinhaClasse até "consultou" uma macro para saber que especialização usar... mas por outro lado, a parametrização em T certifica-se de que a classe não será compilada no executável a não ser que seja utilizada, e simultaneamente, quando esta for compilada, a implementação propícia ao sistema será escolhida. Mas pensemos ainda em melhores exemplos do uso de parâmetros que não são tipos: e se eu tivesse por exemplo em mente uma classe de buffer genérico, que faz conversão automática, em função do tamanho do tipo de variável que recebe? Poderíamos por exemplo, ter uma classe que utiliza uma sintaxe semelhante a algo como OMeuBuffer<char,sizeof(double)>.

A propósito deste último exemplo, esta técnica, de tornar classes transparentes aos tipos utilizados tem um nome: traits. Normalmente, uma trait envolve uma conversão ou um typedef para tornar o código do programador de uma classe completamente agnóstico dos tipos utilizados e devolvidos. Outro conceito útil de metaprogramação é o de policies. Similarmente a uma trait, uma policy filtra o código em função dos tipos utilizados, mas neste caso, faz uma decisão de que algoritmo usar. Imaginemos o exemplo seguinte que refina a nossa anterior classe Sequencia com uma policy:

```
template<typename T, bool
Is_Small=true>
struct EscolheAlgoritmo
{
    static void sort(T *p)
    {
        ... // Fazer um InsertionSort
    }
};

template<typename T>
struct EscolheAlgoritmo<T,false>
{
    static void sort(T *p)
    {
        ... // Fazer um QuickSort
    }
};

template<int N, class T = double>
class OutraSequencia
{
private:
  T *dados;
  ...
public:
  void sort()
  {

EscolheAlgoritmo<T,(N<20)>::sort(dados)
;
  }
  ...
};

...

// Depois, é uma questão de usar com:
OutraSequencia a<15,double>;
OutraSequencia b<30,char>;
a.sort(); // Faz um InsertionSort
b.sort(); // Faz um QuickSort
```

Note-se que, se só se tivesse utilizado a primeira ou a segunda instância de OutraSequencia, um dos algoritmos teria sido excluido da própria compilação. Neste caso, o interessante é ver como uma classe se pode assim adaptar a diversas condições, sem interferir no código do programador. É claro que é fácil imaginar exemplos muito melhores, como por exemplo, aplicações em singletons, mas a ideia é tornar claro como os templates são uma valiosa ferramenta para construir bibliotecas, da qual é um bom exemplo a Standard Template Library. Algumas aplicações avançadas de templates fazem parte da implementação de apontadores inteligentes (tradicionalmente, este tipo de classe tem uma designação com o sufixo _ptr). Apontadores

inteligentes são apontadores que fazem a sua própria gestão de memória, e como tal, utilizam combinações de traits e policies para determinar o seu comportamento em determinadas situações. Desta forma, ajudam a impedir bastantes erros subtis no código. Utilizando templates, há diferentes tipos de ponteiros inteligentes: alguns proíbem a cópia, outros incluem técnicas de reference counting. Os apontadores podem ser origem de muitas dores de cabeça, e toda a ajuda é pouca!

Uma grande porção das aplicações de templates cai na área da computação numérica. Em cálculo numérico, os templates ajudam a reduzir redundâncias de código, minimizando chamadas a métodos virtuais e removendo ambiguidades (através do célebre Barton-Nackman trick, em que uma classe base é recursivamente parametrizada em função das derivadas). No que diz respeito a cálculo, o exemplo mais interessante, contudo, são os templates de expressões. Estes requerem alguma prática e envolvem alguma complexidade. Como tal, é mais razoável dar apenas um curto exemplo do que templates de expressões fazem sem entrar em detalhes, antes de concluir esta discussão.

Imaginemos, portanto, uma típica classe de vector, e um operator+ global que soma dois objectos da mesma classe usando um ciclo for.

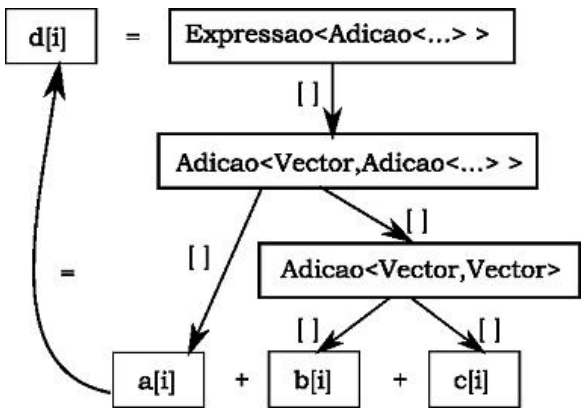
```
template<typename T>
class Vector
{
   ...
};

template<typename T>
const Vector<T>& operator+
  (const Vector<T> &a,
   const Vector<T> &b)
{
  const int s = a.size();
  Vector temp(s);
  for(int i = 0; i < s; ++i)
    a[i] + b[i];
  return temp;
}

...

Vector<double> a,b,c,d;
...
d = a + b + c;  // usando o operador
...
for(int i = 0; i < d.size(); ++i)    //
metodo "a la C"
  d[i] = a[i] + b[i] + c[i];
```

O que se passa com este código é que, enquanto o operador torna a sintaxe agradável ao programador, esta operação é extremamente ineficiente! A primeira alternativa corre pelo menos três ciclos for e cria pelo menos dois vectores temporários. Torna-se preferível escrever o ciclo explicitamente! Ou seja, seria interessante convencer o compilador a gerar um único ciclo for, com um único vector temporário na pior das hipóteses, independentemente do número de termos somados. Vejamos esta solução através da aplicação de templates...

```
template<typename T>
class Vector
{
  private:
    T *_ptr;
    int _num;
  public:
  ...
    template<typename Expr>
    const Vector& operator= (const
Expressao<T,Expr> &expr)
    {
      for(int i = 0; i < _num; ++i) //
ciclo de acesso indirecto
        _ptr[i] = expr[i];
    }
};

...

template<typename T, typename Expr>
class Expressao
{
  private:
    Expr e;
  public:
    inline const T& operator[](int i)
// Membro acedido a partir de Vector
    { return e[i]; }
};

template<typename T, typename OpEsq,
typename OpDir> // Os operandos estão
parametrizados
class Adicao
{
  private:
    OpEsq esq;
    OpDir dir;
  public:
    inline const T& operator[](int i)
    { return esq[i]+dir[i]; }
};

...
```

```
Vector<double> a,b,c,d;
...
d = a + b + c;  // Um único ciclo
```

Lendo de baixo para cima, podemos ver que o compilador começa a interpretar a expressão pela esquerda devido à igual precedência. O desenrolar iterativo à medida que os templates são instanciados é ilustrado na figura seguinte.
d é afectado por um objecto Expressao, consistindo de um



objecto Adicao. Este objecto Adicao, por sua vez, é instanciado com um operando Vector a e com outro objecto Adicao (que representa a soma b+c). Se lermos a definição de Adicao, vemos que tem um operator[], tal como supomos que Vector tem. A soma é resolvida em três passagens de referência: o compilador passa o acesso aos elementos das Vectors ao objecto Expressao (através do seu próprio operator[]). Finalmente, o operator= passa o resultado a d. Deste modo, a soma foi efectuada dentro de um único ciclo for. Obviamente, o processo cria objectos intermédios de classes auxiliares para passar as referências aos elementos dos vectores, mas permite efectuar de forma eficiente uma soma num vector longo, e tudo isto sem sacrificar o conforto de uma sintaxe intuitiva. Esta técnica de implementação de objectos eficientes para computação numérica foi inventada independentemente por Todd Veldhuizen e Daveed Vandevoorde.

Conclusões

Ficou muita coisa por explorar, mas com certeza a ideia das interessantes aplicações de templates no desenho de bibliotecas ficou patente através dos exemplos dados. Os templates de C++ têm uma capacidade impressionante de permitir polimorfismo estático, determinado aquando da compilação, ao ponto de permitir liberdades extremas de compilação condicional. Também permitem construir autênticos idiomas alternativos da linguagem, e tudo isto com um impacto mínimo na eficiência no código compilado. Infelizmente, actualmente este poder sobre a linguagem vem com um custo associado em termos de complexidade, levando a complexidades extremas só para permitir definições intuitivas que tornem classes parametrizadas úteis. Argumentativamente, esta dificuldade só é sentida por quem desenha as bibliotecas para benefício dos programadores que as utilizam, mas erros de compilação difíceis de ler, que se obtêm quando se usam determinados parâmetros em erro, nunca ajudaram ninguém. Obviamente, há soluções que estão a ser implementadas para melhorar esta situação (inclusivamente tornando alguns dos truques aqui mencionados obsoletos), mas isso será abordado num futuro artigo sobre a nova especificação de C++: o C++0x.

Bibliografia

•Ulrich W. Eisenecker: Generative Programming: Methods, Tools, and Applications, Addison-Wesley

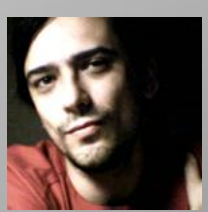
•David Vandervoorde, Nicolai M. Josuttis: C++ Templates: The Complete Guide, Addison-Wesley

•http://ubiety.uwaterloo.ca/~tveldhui/papers/Expression-Templates/exprtmpl.html

•Todd Veldhuizen: Using C++ template metaprograms, C++ Report, Vol. 7, No. 4 (May 1995), pp. 36-43

•John J. Barton, Lee R. Nackman: Scientific and Engineering C++: An Introduction with Advanced Techniques and Examples, Addison-Wesley

SOBRE O AUTOR



Francisco Almeida foi Licenciado em Engenharia Física pela Universidade de Lisboa em 2003 e é agora um aluno finalista de Doutoramento de Física pela Universidade Católica de Lovaina, Bélgica. Para além de Ciência, tem interesse em Programação Orientada por Objectos e Algoritmia. A sua linguagem de programação preferida sempre foi o C++, mas também considera Java, C# e D muito interessantes.

francisco.almeida@portugal-a-programar.org

Francisco Almeida

Processamento de texto em AWK - Parte II

Introdução

O AWK é uma linguagem utilizada em ambientes UNIX para processar dados baseados em texto. Na edição anterior apresentámos aqui um artigo de introdução a esta linguagem. Nesta edição, apresentamos alguns dos seus aspectos mais avançados.

Campos e variáveis

Nos exemplos anteriores, os acessos a campos eram sempre feitos através de um literal inteiro (do tipo `$1`). No entanto, este acesso também pode ser feito através de um inteiro armazenado numa variável.

Por exemplo, o seguinte programa lê um número no primeiro campo e escreve o seu conteúdo:

```
{ campo = $1; print $campo }
```

Para além disto, os campos (tal como as variáveis especiais) podem ser modificados como se fossem variáveis. Por exemplo, o seguinte programa escreve o input sem o segundo campo:

```
{ $2 = ""; print }
```

Arrays

Em AWK é possível definir arrays (apenas unidimensionais) para guardar um conjunto de valores relacionados. Apesar de apresentarem uma sintaxe semelhante aos arrays utilizados em C, existem algumas diferenças quanto à sua forma de utilização:

- Não é necessário especificar uma dimensão para o array. Ele é ajustado automaticamente sempre que é necessário guardar um novo valor;
- Os arrays são associativos, ou seja, funcionam como um mapa entre uma chave (o índice) e um valor.

O seguinte exemplo ilustra as vantagens da utilização deste tipo de arrays, contando as palavras (registos) no input:

```
{ for(i = 1; i <= NF; i++)
contador[$i]++ } # O ciclo é executado
para cada linha de input.
END { for(palavra in contador) print
palavra, contador[palavra] }
```

A sintaxe do ciclo for na última linha permite iterar sobre todos os elementos de um array.

Este ciclo for no final corresponde à iteração sobre o array. Note-se que este ciclo não garante a ordem no acesso às posições do array. Note-se que este ciclo não garante a nenhuma ordem no acesso aos dados.

A instrução delete pode ser utilizada para remover um elemento do array.

Funções

As funções tem como objectivo simplificar a resolução de problemas que de outra forma se tornariam complexos. Como há funções às quais é necessário recorrer com alguma frequência, existem um conjunto de funções predefinidas e prontos a ser utilizadas.

Funções sobre números

- sqrt(n): Devolve a raiz quadrada de n;
- log(n): Devolve o logaritmo de base e de n;
- exp(n): Devolve a potência de base e de n;
- int(n): Devolve a parte inteira de n.

Exemplos:

```
sqrt(4) # Devolve 2
sqrt(5) # Devolve 2,23607
log(1) # Devolve 0
log(2) # Devolve 0,693147
exp(0) # Devolve 1
exp(1) # Devolve 1,71828
int(3.14159) # Devolve 3
```

Funções sobre strings

• substr(string, inicio, tamanho): Devolve a substring de string que começa em inicio e tem o tamanho tamanho.
• split(string, array, separador): Guarda em array as substrings de string separadas pela string separador. É usado um espaço como separador se este for omitido.
• index(string, padrao): Devolve o índice da primeira ocorrência de padrao em string (ou o se não existir nenhuma ocorrência).

Nota: O ínicio da string corresponde ao índice 1. Da mesma forma, a função split coloca as substrings em índices a partir de 1.

Exemplos:

```
substr("programar", 4, 4) # Devolve
"gram"
split("portugal-a-programar",
palavras, "-")
print palavras[1] # portugal
print palavras[2] # a
print palavras[3] # programar
index("portugal-a-programar", "port")
# Devolve 1
index("portugal-a-programar", "b") #
Devolve 0
```

"printf" e "sprintf"

Tem sido usada a instrução print para escrever texto no ecrã. No entanto, também é possível usar a função printf para produzir uma string formatada.

A formatação através da função printf funciona da mesma forma que em C, utilizando (principalmente) os seguintes códigos de controlo de formato:

• %c: Escreve um char;
• %d: Escreve um int em formato decimal;
• %f: Escreve um float;
• %o: Escreve um número em formato octal;
• %s: Escreve uma string;
• %u: Escreve um unsigned int;
• %x: Escreve um int em formato hexadecimal.

Com isto, é possível escrever o número de cada registo do input em formatos decimal e hexadecimal com o seguinte programa:

```
{ printf("%d %x\n", NR, NR) }
```

Será produzido o seguinte texto:

```
1 1
2 2
3 3
4 4
5 5
6 6
7 7
8 8
9 9
10 a
11 b
12 c
13 d
14 e
15 f
16 10
17 11
18 12
19 13
20 14
```

A função sprintf funciona de forma análoga, mas devolve a string em vez de a escrever.

Funções definidas pelo utilizador

Para além das funções predefinidas, é possível também definir novas funções. Estas funções devem ser definidas no início do programa (antes do primeiro bloco). Para o fazer utiliza-se a seguinte sintaxe:

```
function nome(parametros) {
  instrucoes
}
```

A instrução return indica qual o valor a ser devolvido pela função quando a sua execução termina.

Redireccionamento de output

Para além de escrever o resultado no ecrã, também é possível enviá-lo para um ficheiro ou comando. Em AWK existem três operadores relacionados com o redireccionamento de output:

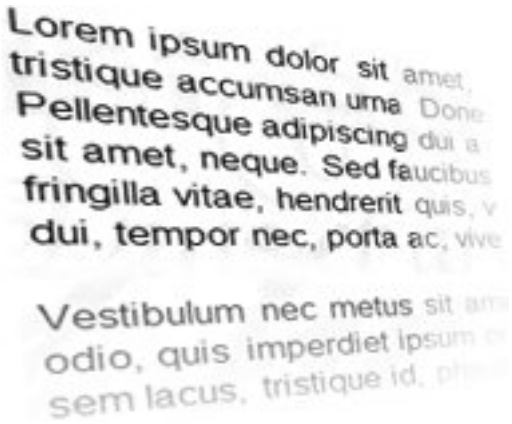
• >: Redirecciona o output para um ficheiro;
• >>: O mesmo que o anterior, mas caso o ficheiro já exista o output é adicionado no fim;
• |: Redirecciona o output para um comando Unix.

Por exemplo, é possível enviar o número de cada registo para um ficheiro out.txt com o seguinte programa:

```
{ print NR >> "out.txt" }
```

Conclusão

Este artigo serve de complemente ao artigo introdutório publicado na edição anterior. Este artigo pode parecer complexo para um programador não-familiarizado nesta linguagem, sendo nesse caso aconselhável a leitura do referido artigo na edição anterior.



SOBRE O AUTOR

Fábio Ferreira frequenta desde 2005 o curso de Engenharia Informática e de Computadores no Instituto Superior Técnico, estando actualmente na área de especialização de Sistemas Inteligentes. Para além disto, tem interesse por algoritmia, Python e sistemas Unix.

fabio.ferreira@portugal-a-programar.org

Fábio Ferreira

DEI@Academy

Apresentação

O DEI Academy é uma plataforma online que tem como objectivo ajudar alunos do secundário a aprender a programar.

Uma iniciativa do Departamento de Engenharia Informática da Universidade de Coimbra, o site ensina principalmente a linguagem Python e muitos dos conceitos que vais aprender em qualquer curso de informática.

Se ainda não sabes se estás interessado em seguir informática, ou simplesmente queres aprender mais sobre programação, este site deverá ser uma boa ferramenta.

O site está disponível no endereço http://academy.dei.uc.pt.

Passado, Presente e Futuro

O site abriu só no ano passado e reúne actualmente mais de 300 utilizadores frequentes.

Melhorámos o aspecto da página, adicionámos mais conteúdos, criámos um sistema de desafios e pontos integrado no site, entre outros.

No futuro próximo pretendemos integrar ainda mais desafios e concursos periódicos.

Desafios

Não se aprende a programar sem praticar muito.

Agora enquanto aprendes os conceitos na página, podes resolver mini-desafios que aparecem directamente na página dos manuais.

Estes mini-desafios são resolvidos na mesma linguagem dos conteúdos (Python) e podem ser submetidos directamente para avaliação automática. O sistema pretende ser muito simples, e evitar o uso de sistemas de avaliação mais complexos.

A resolução dos desafios dá-te direito a pontos sempre que a resposta seja aceite. No futuro existirá um ranking dos utilizadores com mais pontos.

Comunidade

Não queremos que os utilizadores estejam limitados aos nossos conteúdos!

Queremos acima de tudo gerar discussão, partilha de conhecimento e sentido de entreajuda. Incentivamos os utilizadores a partilhar todo o género de informação e feedback ao site no nosso fórum.

Sistema de Avaliação Automática

Para aqueles mais interessados no funcionamento interno da nossa plataforma, o sistema de avaliação automática do código Python deverá ser a funcionalidade mais interessante.

Actualmente este sistema é feito através do serviço Google App Engine que permite executar código Python de forma escalável, sem existir preocupações com o possível excesso de carga no processador, e isolar cada código para motivos de segurança.

Conclusão

Espero ter dado a conhecer melhor o projecto, e que tenha despertado a curiosidade dos leitores,
http://academy.dei.uc.pt.

Fábio Pedrosa

Google Web Toolkit

Após a introdução à plataforma Google Web Toolkit (GWT) na anterior edição da Revista Programar (19ª Edição: http://blog.portugal-a-programar.org/2009/04/14/revista-programar-19ª-edicao-abril-2009/) eis um novo artigo numa nova edição da Revista, sobre a mesma plataforma, mas com novos conceitos e algumas novidades.

Nota: Se ainda não leu o primeiro artigo de Introdução ao Google Web Toolkit, anteriormente referido, é recomendado que o faça para que deste modo possa se familiarizar com a plataforma, e assim, tirar o máximo proveito deste novo artigo.

Resumindo, neste artigo…

Irão estar em destaque diversos aspectos do Google Web Toolkit. Inicialmente será abordado o novo plugin do GWT para Eclipse, lançado recentemente pela Google e que permite a integração de aplicações GWT com o Google Application Engine (GAE) . De seguida é demonstrada uma das mais importantes peças do GWT, o mecanismo RPC, que permite a fácil implementação de mecanismos de comunicação entre cliente e servidor.

O que vai aprender?

Ao longo do artigo o leitor irá:

- Configurar o Google Web Toolkit no Eclipse, utilizando para tal o novo plugin disponibilizado pela Google;
- Implementar mecanismos de comunicação entre cliente/servidor através de RPC (Remote Procedure Call);

Seguindo estes passos, no final do artigo o leitor terá construído uma simples aplicação GWT, resultado da configuração e implementação dos aspectos anteriormente referidos, compreendendo não só como elas funcionam como também como é que estas se integram na plataforma GWT.

Requisitos

Para a realização deste artigo foram utilizadas as seguintes tecnologias:

- Eclipse 3.4.2 (Ganymede). De acordo com a documentação da Google (http://code.google.com/intl/pt-PT/eclipse/docs/download.html), o plugin funciona com as versões Eclipse 3.3(Europa) e Eclipse 3.4(Ganymede).
- GWT 1.6;
- Google Chrome.
- Sistema Operativo XP Home Edition

Nota: Note-se que para este exemplo é utilizado o Eclipse, mas o desenvolvimento de aplicações GWT pode ser feito utilizando qualquer IDE Java.

"Google Plugin for Eclipse"

Recentemente a Google anunciou o lançamento de um novo plugin para o Eclipse. Este novo plugin, além de facilitar a criação de aplicações GWT, permite a integração das mesmas no Google App Engine (o que é o Google App Engine? - http://code.google.com/intl/pt-PT/appengine/docs/whatisgoogleappengine.html). Usando uma simples sequência de tarefas é possível publicar qualquer aplicação GWT nos servidores da Google.

Esta é realmente uma grande novidade, pois inicialmente o Google App Engine apenas permitia o desenvolvimento de aplicações em Python e só muito recentemente foi anunciado o suporte à linguagem Java. No entanto, é necessário ter alguns cuidados na medida em que a linguagem Java não é inteiramente suportada. As limitações estão documentadas em http://groups.google.com/group/google-appengine-java/web/will-it-play-in-app-engine.

Este novo plugin da Google para o GWT inclui wizards para a criação de entrypoints, módulos e páginas Web, suporte à realização de testes unitários, entre outros.

Instalando o Plugin

É possível fazer o download do Eclipse no seguinte endereço- http://mac.softpedia.com/get/Development/Editors/Eclipse-SDK.shtml.
Depois de descarregado o Eclipse, é possível então proceder-se à instalação do plugin. Para quem não está muito familiarizado com a plataforma, o Eclipse torna a instalação de plugins uma tarefa bem simples, na medida em que esta pode ser feita utilizando apenas um url que indica o plugin a instalar. Os passos para a instalação são mostrados de seguida.

Depois de abrir o Eclipse, ir ao menu Help -> Software Updates

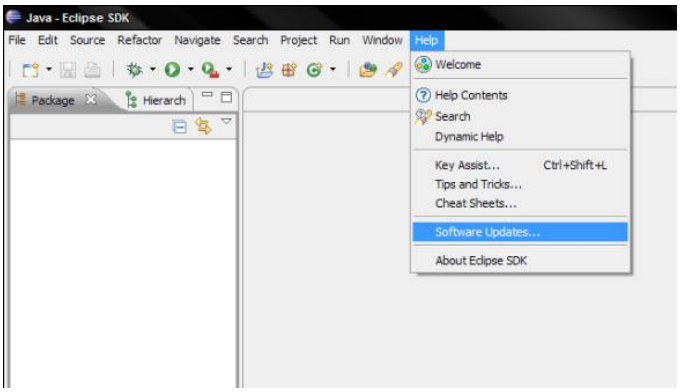


Figura 1 – Eclipse – Instalando Plugins

Logo de seguida, surge uma nova janela designada "Software Updates and Add-ons" onde é possível verificar actualizações e software disponível. Clicando no botão à direita "Add Site" surge novamente uma nova janela onde é necessário colocar o url do plugin desejado. Neste caso, para a instalação do plugin Google, o url é o seguinte - http://dl.google.com/eclipse/plugin/3.4.

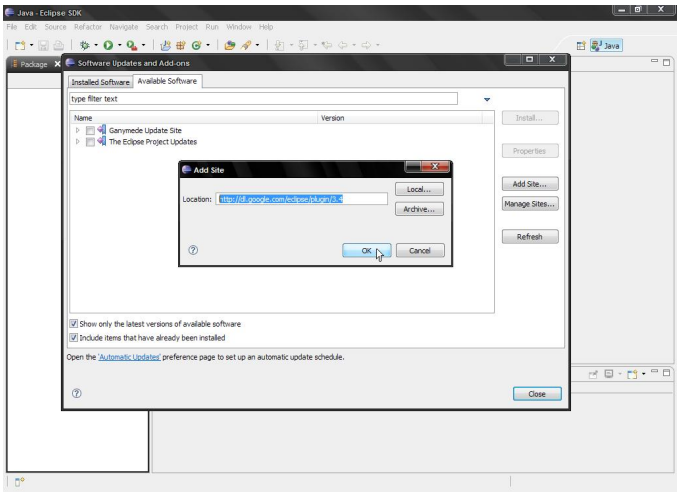


Figura 2 – Fornecendo o endereço do plugin

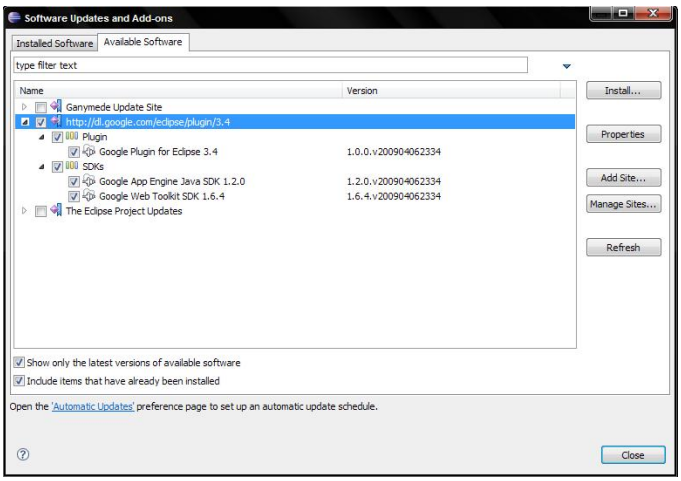


Figura 3 – Lista de Plugins

O plugin é então adicionado à lista de plugins. Como é possível verificar na imagem seguinte, são instalados não só o plugin, como também os SDK's do Google App Engine e a ultima versão da plataforma Google Web Toolkit, o GWT 1.6.

Figura 4 – Instalação do plugin

Para começar a instalação, seleccionar o checkbox correspondente e clicar no botão "Install" à direita. Inicia-se então o processo de instalação do plugin no Eclipse. Finalizado o processo, o plugin está instalado, configurado e pronto a ser utilizado.

Criando uma aplicação GWT

A partir de agora é possível criar aplicações GWT no Eclipse apenas com uns simples cliques. Para criar uma nova aplicação GWT, basta clicar com o botão direito do rato sobre a área à esquerda do Eclipse no "Package Explorer". Irá surgir um menu onde deverá clicar em "New", aparecendo assim um submenu, no qual deverá clicar em "Web Application Project", tal como mostra a figura seguinte.

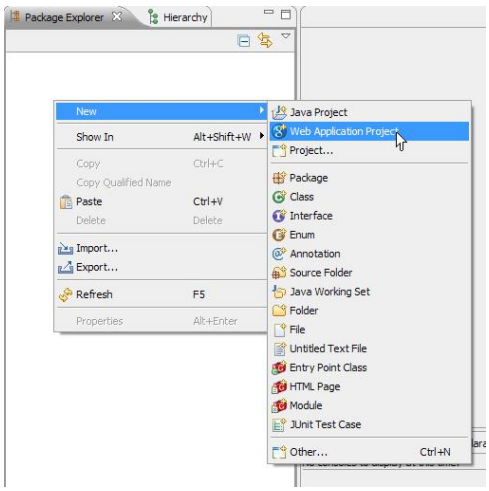


Figura 5 – Criando uma aplicação GWT no Eclipse

Irá surgir uma nova Janela onde devem ser especificados o Nome do Projecto e o package onde os ficheiros do projecto serão alojados. Vamos definir o Nome do Projecto como "MyFirstGWTApp" e o package como "gwt.sample.myproject".
Clicando em "Finish" é gerada toda a estrutura do projecto.

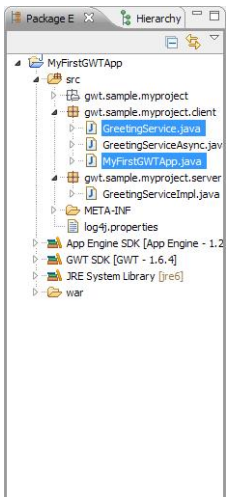


Figura 6 – Aplicação GWT

Foram criados um conjunto de ficheiros padrão, onde neste caso, o "MyFirstGWTApp.java" consiste no EntryPoint da aplicação e os restantes ficheiros implementam o serviço RPC de comunicação entre o cliente e o servidor. Temos assim uma aplicação GWT por omissão, pelo que a partir desta estrutura inicial é possível então desenvolver a aplicação Web desejada.

Para correr a aplicação, basta clicar com o botão direito do rato em cima do projecto, clicando depois em Run As -> Web Application, executando a aplicação em Hosted Mode. Hosted Mode consiste no browser interno do GWT usado na fase de desenvolvimento, permitindo o debbuging das aplicações em código Java. Para mais informação pode consultar o artigo de introdução ao GWT publicado na 19ª edição da Revista (http://blog.portugal-a-programar.org/2009/04/14/revista-programar-19ª-edicao-abril-2009/).

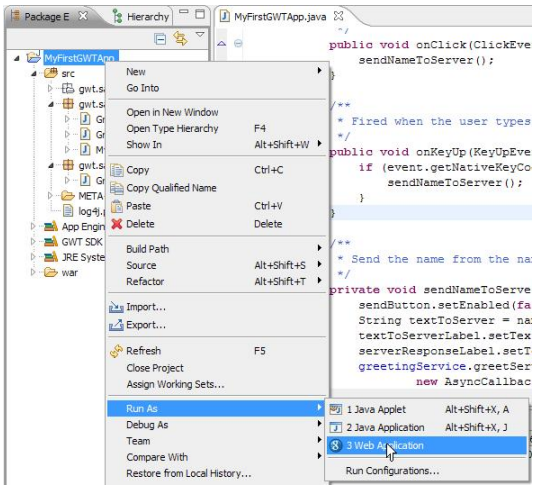


Figura 7 – Executando a Aplicação

Depois de executada a aplicação, é possível visualizar na consola do Eclipse a seguinte mensagem "The server is running at http://localhost:8080", o que significa que a aplicação está também em execução no servidor local, pelo que pode ser acedida através do browser pelo url referido.
Deste modo, o plugin da Google torna muito simples a criação de aplicações GWT.

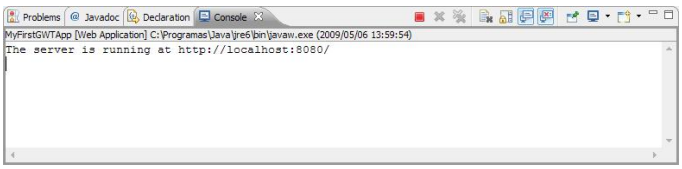


Figura 8 – Consola do Eclipse

Disponibilizando a sua aplicação Web…

Como já foi dito anteriormente, este plugin também permite a integração de aplicações GWT no GAE. Ou seja, fazendo uso da infra-estrutura da Google, é possível desenvolver uma aplicação Web no Eclipse e, em poucos passos, publicá-la nos seus servidores.
Como pode ser observado, ao instalar o plugin no Eclipse surgiram novos elementos no menu superior, sendo um destes o logótipo do GAE.

Figura 9 – Publicando a aplicação no Google App Engine

Ao clicar neste botão é possível publicar a aplicação nos servidores da Google. Mas para isso é necessário, antes de mais, criar uma conta no GAE. Este passo não é aqui detalhado mas é possível consultar os passos necessários para tal no seguinte link - http://code.google.com/intl/pt-PT/appengine/docs/java/gettingstarted/uploading.html.
Depois de criada a conta no GAE e o ID para a respectiva aplicação, basta clicar no botão acima referido, iniciando-se então a publicação da aplicação.

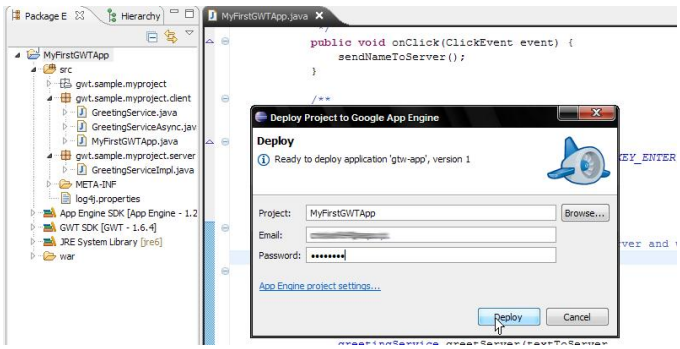


Figura 10 – Publicando a aplicação no Google App Engine (2)

Depois de publicada, é possível aceder à aplicação através do domínio criado no GAE

RPC no GWT – "Hello Server! Can I ask you something?"

O que é RPC?

O Remote Procedure Calls (RPC) – Chamadas de Procedimento Remoto) é um mecanismo que permite fazer a chamada a procedimentos remotos como se estes fossem locais. Ou seja, implementa um serviço de comunicação entre sistemas remotos, abstraindo o programador dos detalhes de implementação.
Assim, sempre que o programador necessite de comunicar com o respectivo serviço remoto, fá-lo de forma transparente como se de uma chamada local se tratasse.
Deste modo, para a comunicação entre cliente/servidor, o GWT disponibiliza o mecanismo RPC fazendo com que cliente e servidor possam trocar/partilhar objectos Java entre si através do protocolo HTTP, de uma forma rápida e transparente.

Como funciona? Tão simples quanto isto...
De seguida é mostrado o diagrama correspondente ao RPC no GWT:

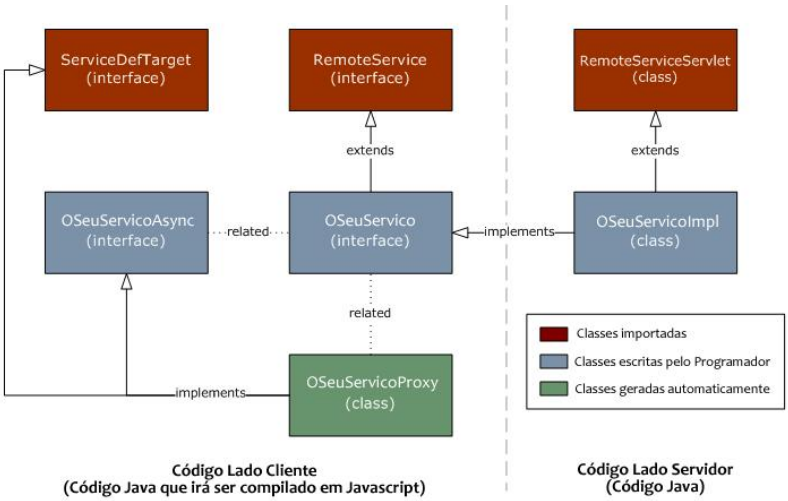


Figura 11 – Diagrama RPC

Como é possível observar no diagrama anterior (retirado e traduzido de http://library.igcar.gov.in/readit2007/tutori/tools/gwt-windows-1.4.10/doc/html/com.google.gwt.doc.DeveloperGuide.RemoteProcedureCalls.PlumbingDiagram.html) existem três cores distintas, vermelho, azul e verde, que representam respectivamente:

- as interfaces disponibilizadas pelo GWT;
- as classes que são necessárias implementar pelo programador;
- e a classe que é gerada automaticamente pelo GWT, ou seja o Proxy, que irá servir de intermediário entre o cliente e o servidor.

As classes de maior importância para o programador são obviamente as pintadas a azul, pois são estas que têm de ser implementadas pelo próprio.

Assim, para se definir um serviço GWT é necessário implementar no lado cliente duas interfaces, o "YourService" e o "YourServiceAsync" e, no lado servidor é então feita a respectiva implementação ("YourServiceImpl") do serviço (interfaces) declarado no lado cliente.

Até agora, o que pode provocar alguma confusão é a interface "YourServiceAsync". Isto significa que para cada serviço declarado, é necessário criar uma versão assíncrona do mesmo. Mas agora a pergunta que se coloca é: Porquê criar uma versão assíncrona do serviço?

AJAX – Asynchronous Javascript and XML

O GWT é uma plataforma orientada ao desenvolvimento de aplicações Web baseada na tecnologia Asynchronous Javascript and XML (AJAX). A grande novidade desta tecnologia é o facto de não serem feitos pedidos de páginas completas ao servidor, mas sim realizados pequenos pedidos assíncronos que ao serem retornados pelo servidor actualizam pequenas partes da página.

Deste modo o GWT, no mecanismo de comunicação com o servidor, ao fazer um pedido no lado cliente tem que assegurar uma forma de poder receber a resposta quando este for enviado pelo servidor. Para isso, é criado um objecto callback do tipo AsynCallback que irá conter a resposta enviada pelo servidor. Podemos pensar no AsyncCallback como um EventListener que fica "à escuta" da resposta do servidor para que quando este retorne o resultado o cliente possa ser notificado. Deste modo a aplicação não fica "bloqueada" à espera da resposta e outras tarefas podem ser realizadas entretanto.

Mais adiante irá ser implementado um exemplo prático onde vai ser possível visualizar e compreender melhor como funciona o AsyncCallback, bem como todo o processo de comunicação cliente/servidor no GWT.

Serialização

De uma forma simples, a serialização consiste em transformar um objecto num conjunto de bits armazenados consecutivamente; e a "deserialização" consiste na transformação inversa, de forma a ser possível recuperar o objecto tal como se encontrava no momento da serialização. Esta técnica é muito útil quando se pretende, por exemplo, guardar objectos num buffer de memória, num ficheiro, ou transmiti-los através da Rede.

Esta técnica é utilizada pelo mecanismo de RPC no GWT, pois como já foi dito anteriormente, o RPC permite a troca de objectos entre cliente e servidor através do protocolo HTTP.

Mas para que tal aconteça é necessário que os objectos a partilhar implementem serialização. Um objecto destinado ao cliente é serializado pelo servidor, e depois enviado através de HTTP; já no lado cliente o objecto é recuperado através da sua "deserialização".

Porquê serialização?

Um objecto representa uma estrutura de dados cuja informação está normalmente "espalhada" pela memória e que pode ser acedida aleatoriamente. No entanto, se pretendemos transmitir essa estrutura através de um canal sequencial, é necessário que essa informação seja agrupada, tarefa essa que é realizada pela serialização. Depois de transmitida, essa informação é novamente "recolocada" na memória recuperando-se a estrutura de dados/objecto até então serializado, ou seja, a chamada deserialização.

Este parece mostrar-se um processo complicado mas, uma vez que a serialização já é suportada pelo GWT, torna-se tudo muito simples e transparente para o programador. Tudo o que este tem de fazer é garantir que os objectos partilhados entre cliente/servidor implementem a serialização.

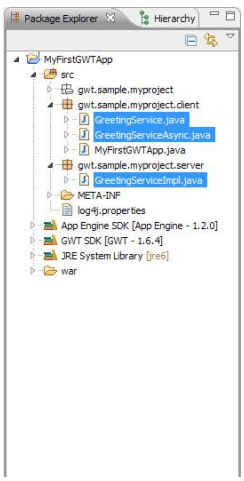
Note-se que falarmos de serialização no GWT não é bem como falar de serialização em Java, pois não podemos esquecer que temos Javascript no lado cliente e Java no lado servidor.
Deste modo, um objecto é serializável no GWT se:

- é um tipo primitivo, como char, byte, short, int, long, boolean, float, or double;
- é String, Date, Character, Byte, Short, Integer, Long, Boolean, Float, or Double;
- é um array de tipos serializáveis;
- é uma classe serializável;
- é uma classe que tem pelo menos uma subclasse serializável.

Para saber mais sobre serialização no GWT consulte http://developerlife.com/tutorials/?p=131.

RPC - Na prática…

Voltando à aplicação GWT "MyFirstGWTApp" criada logo no início deste artigo, é possível observar que já é implementado automaticamente um serviço para esse efeito, o "GreetingService". Este é um serviço básico, com uma simples troca de mensagens entre o cliente e o servidor, mas que já nos dá uma ideia de como são implementados os mecanismos de comunicação cliente/servidor no GWT.



Pela imagem observa-se que são implementados os três passos necessários para estabelecer a comunicação entre cliente e servidor. No lado cliente é declarado o serviço GreetingService e a sua "versão assíncrona" GreetingServiceAsync e no lado servidor a implementação do respectivo serviço, designado GreetingServiceImpl.

Este exemplo será assim reaproveitado para que o leitor possa entender na prática como as coisas se processam.
No entanto, neste caso, o cliente e o servidor trocam entre si strings, que consistem em objectos da classe String, que por sua vez já implementa serialização. Na tentativa de explorar um pouco mais a serialização e a troca de objectos entre o cliente e o servidor, vamos criar uma classe GreetingMessage (que implemente serialização) de modo a que a mensagem retornada pelo servidor seja instância da respectiva classe. Deste modo o leitor aprenderá a implementar a serialização e a utilizá-la sempre que precisar partilhar objectos entre cliente/servidor.

Implementando o mecanismo RPC

A nossa aplicação, designada MyFirstApp, é constituída por 5 ficheiros:

No lado Cliente:

- MyFirstApp (consiste no EntryPoint da aplicação)

GreetingService (corresponde ao serviço requisitado pelo cliente)

- GreetingServiceAsync (versão assíncrona do serviço);
- GreetingMessage (classe que será implementada por nós para a "construção" de mensagens – a trocar entre cliente/servidor);

No lado Servidor:

- GreetingServiceImpl (corresponde a implementação do serviço declarado no lado cliente)

Antes de mais... Implementando a classe GreetingMessage

Deverá criar antes de mais uma nova classe,no lado cliente, designada Greeting Message e copiar o seguinte código para esta nova classe:

```
//Greating Message.java
package gwt.sample.myproject.client;

import
com.google.gwt.user.client.rpc.IsSerial
izable;

public class GreetingMessage
implements IsSerializable {

    private static final long
serialVersionUID = 1L;

    //attributes
     private String title;
     private String content;

     //empty constructor --> is
mandatory for serializable
classes...
     public GreetingMessage()   {}

    //constructor
    public GreetingMessage(String
title, String content)
    {
        this.title   = title;
        this.content = content;

    }

    //get's and set's methods

    public String getTitle()
    {
        return title;
    }

    public String getContent()
    {
        return content;
    }

    public void setTitle(String
newTitle)
    {
        this.title = newTitle;
    }

    public void setContent(String
newContent)
    {
        this.content = newContent;
    }

     //creating a Message
    public GreetingMessage
createNewMessage(String title, String
content)
    {
        GreetingMessage newMessage =
new GreetingMessage(title, content);

        return newMessage;
    }
}
```

Como podemos observar, e tal como já foi dito anteriormente, as classes cujos objectos irão ser partilhados entre cliente e servidor devem implementar serialização, mais concretamente a interface IsSerializable. Um aspecto que ainda não foi referido é o facto de que todas as classes que implementem serialização têm de possuir um construtor vazio. É definido que a classe GreetingMessage possui como atributos title e content, ambos do tipo String, que irão consistir respectivamente no título e no conteúdo da mensagem a criar.

Implementando o Serviço GreetingService - 1º Passo

Uma vez definida a classe GreetingMessage será agora implementado o respectivo serviço GreetingService. O primeiro passo para a elaboração do serviço é a definição da sua interface no lado cliente. A interface tem que estender a interface RemoteService e, tanto os parâmetros como os valores de retorno do serviço (método) têm de ser serializáveis. Deste modo temos:

```
package gwt.sample.myproject.client;

import com.google.gwt.user.client.rpc.RemoteService;
import com.google.gwt.user.client.rpc.RemoteServiceRelativePath;

/**
 * The client side stub for the RPC service.
 */
@RemoteServiceRelativePath("greet")
public interface GreetingService extends RemoteService {

    public GreetingMessage greetServer(String name);
}
```

Figura 13 – Definindo um serviço no GWT

Temos assim a interface (serviço) GreetingService que estende a interface RemoteService e que implementa um método de mesmo nome "GreetingService", que tem como parâmetro uma String que corresponde ao nome de cliente, e que retorna uma instância da classe GreetingMessage por sua vez serializável.

Implementando o Serviço GreetingService - 2º Passo

Agora que o serviço já foi definido, vamos definir a sua versão assíncrona.
As classes que representam a versão assíncrona do serviço têm de ter sempre nome igual à classe que define o serviço seguido do sufixo "Async". Deste modo, para este caso a versão assíncrona será designada "GreetingServiceAsync".

```
package gwt.sample.myproject.client;

import com.google.gwt.user.client.rpc.AsyncCallback;

/**
 * The async counterpart of <code>GreetingService</code>.
 */
public interface GreetingServiceAsync {
    public void greetServer(String input, AsyncCallback<GreetingMessage> callback);
}
```

Figura 14 – Versão assíncrona do serviço

O método definido pela classe assíncrona não deve possuir valor de retorno, ou seja, deve ser sempre "void" uma vez que o resultado retornado pelo servidor irá ser guardado num objecto callback (já referido anteriormente), objecto este que deve ser adicionado como parâmetro no método da classe assíncrona, tal como mostra a imagem anterior. Relembrando: quando o servidor responde, o objecto callback guarda a resposta e notifica o cliente.

Implementando o Serviço GreetingService - 3º Passo

Finalmente, para termos uma completa definição do serviço falta somente a sua respectiva implementação no lado servidor. Tal como acontece com a versão assíncrona do serviço, a classe que o implementa deve ter sempre nome igual seguido do sufixo "Impl". Deste modo, para o nosso exemplo, a classe que implementa o serviço terá o nome "GreetingServiceImpl".
Além disso, esta classe tem de estender a interface "RemoteServiceServlet". Esta interface representa a classe base servlet e tem como função processar os pedidos do cliente. Deste modo, para cada serviço implementado é criado um servlet (o que é um servlet? - http://pt.wikipedia.org/wiki/Servlet) que se irá responsabilizar pelo processamento do pedido. Mais adiante será explicado o que é um servlet e como este é criado e configurado.

De seguida, deve substituir o código existente no ficheiro "GreetingServiceImpl.java" pelo seguinte código:

```
package gwt.sample.myproject.server;
import
gwt.sample.myproject.client.GreetingSer
vice;
import
gwt.sample.myproject.client.GreetingMes
sage;
import
com.google.gwt.user.server.rpc.RemoteSe
rviceServlet;

/**
 * The server side implementation of
the RPC service.
 */
@SuppressWarnings("serial")
public class GreetingServiceImpl
extends RemoteServiceServlet
implements
                GreetingService {
    GreetingMessage gMessagePointer =
new GreetingMessage();
    public GreetingMessage
greetServer(String input) {
```

```
        //server build new greeting
message and returns to client...
        return buildMessage(input);
    }

    //building new Greeting Message...
    private GreetingMessage
buildMessage(String input) {
        String serverInfo =
getServletContext().getServerInfo();
        String userAgent   =
getThreadLocalRequest().getHeader("User
-Agent");
        String content = "Hello, " +
input + "!! Welcome to GWT World"
                          +"!<br><br>I
am running " + serverInfo
                     + ".<br><br>It
looks like you are using:<br>" +
userAgent;
        //create the message....
delegates to GreetingMessage class
GreetingMessage gMessage =
gMessagePointer.createNewMessage("",
content);
        return gMessage;
    }
}
```

Como pode ser observado foram feitas algumas alterações ao código que já se encontrava criado... A diferença é que o servidor agora utiliza a classe GreetingMessage para construir a mensagem de resposta ao cliente. O resultado acaba por ser o mesmo, no entanto o servidor envia a mensagem como um objecto serializado, instância da classe GreetingMessage anteriormente criada por nós.

RPC - Mission acomplished?... Quase... - 4ºPasso

Como já foi dito, para cada serviço implementado é criado um servlet responsável por suportar o respectivo serviço, ou seja, ao criar um novo serviço, o programador tem de mapear ou configurar um novo servlet ao respectivo serviço. Mas, como configurar um novo servlet? É muito simples...
A aplicação MyFirstApp contém um ficheiro designado web.xml (MyFirstApp/war/WEB-INF/web.xml) onde são configurados os servlets.

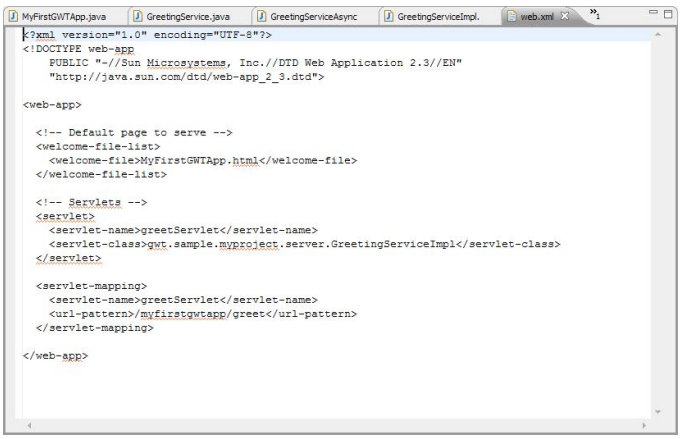


Figura 15 – web.xml

Desta forma, para definir/configurar um novo servlet é necessário definir:

- o seu nome;
- a classe que implementa o serviço; e
- o endereço onde é o serviço é prestado, ou seja, o endereço do servidor que contém o ficheiro que implementa o serviço.

Na imagem anterior, se tomarmos atenção ao url do serviço, vemos que este é definido de uma forma muito simples (/myfirstgwtapp/greet); o GWT vem mais uma vez facilitar a vida do programador, na medida em que utiliza uma anotação para automatizar o processo de criação do endereço.

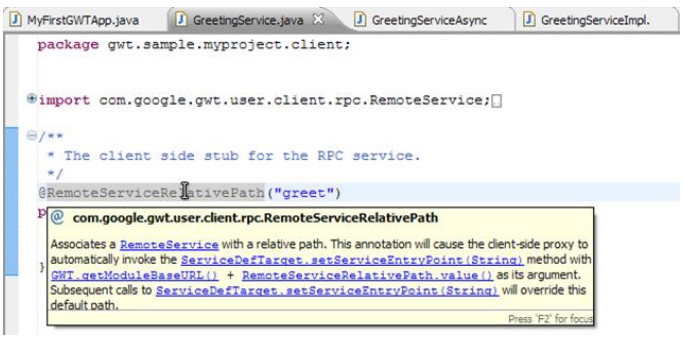


Figura 16 – Definindo o endereço

A interface GreetingService, interface do serviço no lado cliente, utiliza uma anotação designada RemoteServiceRelativePath que irá fazer com que o proxy, gerado automaticamente pelo GWT, realize as configurações necessárias para o mapeamento do serviço. Note-se que a anotação tem como argumento um nome igual àquele que foi definido no endereço do servlet do ficheiro web.xml - /myfirstgwtapp/**greet**. Estes nomes terão de ser sempre iguais.

E... voilá... o serviço está completamente implementado, configurado e pronto a ser usado.

5º e último passo – Invocando o serviço GreetingService

Agora que o serviço já está disponível, podemos então invocá-lo na nossa aplicação. Este é invocado na classe principal da aplicação, ou seja, na classe MyFirstGWTApp que corresponde ao EntryPoint da mesma.

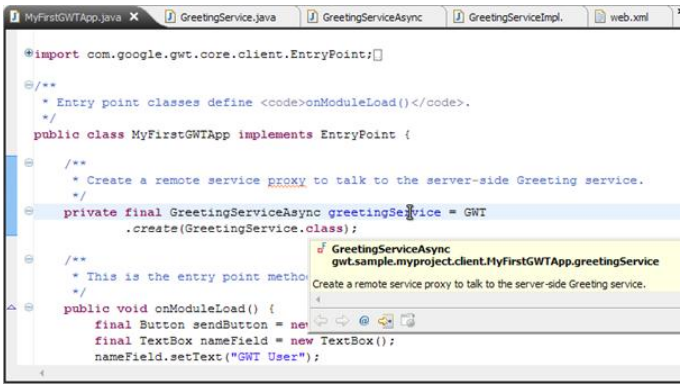


Figura 17 – Remote Proxy

Olhando para a classe MyFirstApp, vemos que logo de início é criada uma instância do serviço que vai ser utilizado. Invocando a função GWT.create() é permitida a criação de um proxy que irá ser o intermediário entre o cliente e o servidor para o respectivo serviço.

Agora que o proxy para o serviço está criado, é então definida a função que irá invocar o serviço. Para este exemplo, o método é designado sendNameToServer().

```
/**
 * Send the name from the nameField to the server and wait for a response.
 */
private void sendNameToServer() {
    sendButton.setEnabled(false);
    String textToServer = nameField.getText();
    textToServerLabel.setText(textToServer);
    serverResponseLabel.setText("");
    greetingService.greetServer(textToServer,
            new AsyncCallback<GreetingMessage>() {
        public void onFailure(Throwable caught) {
            // Show the RPC error message to the user
            dialogBox
                    .setText("Remote Procedure Call - Failure");
            serverResponseLabel
                    .addStyleName("serverResponseLabelError");
            serverResponseLabel.setHTML(SERVER_ERROR);
            dialogBox.center();
            closeButton.setFocus(true);
        }

        public void onSuccess(GreetingMessage result) {
            dialogBox.setText("Remote Procedure Call");
            serverResponseLabel
                    .removeStyleName("serverResponseLabelError");

            serverResponseLabel.setHTML(ToString(result));
            dialogBox.center();
            closeButton.setFocus(true);
```

Figura 18 – Invocando o serviço GreetingService

Utilizando a instância do serviço anteriormente criada (greetingService), é então invocado o método assíncrono do serviço (pintado a azul), passando como argumentos o nome do cliente textToServer e o objecto callback responsável por notificar o cliente aquando da resposta do servidor.

onFailure, onSucess

Sabe-se que, por vezes, não é possível estabelecer comunicação com o servidor pelas mais variadas razões, pelo que o pedido do cliente pode ter dois resultados possíveis: sucesso e insucesso. É por isso que o objecto callback suporta dois métodos, designados onFailure e onSucess, onde o programador define o desejado para cada um dos casos anteriores.

O método onFailure() recebe como parâmetro a causa que provocou a falha de comunicação com o servidor, enquanto que o método onSucess() recebe como parâmetro o resultado enviado pelo servidor, consequência do pedido do cliente.

Executando a aplicação

A aplicação MyFirstGWT está pronta a ser executada. Temos deste modo uma aplicação que implementa, através de RPC, um serviço designado GreetingService, que por sua vez permite a troca de uma mensagem entre o cliente e o servidor.
O resultado é o seguinte:

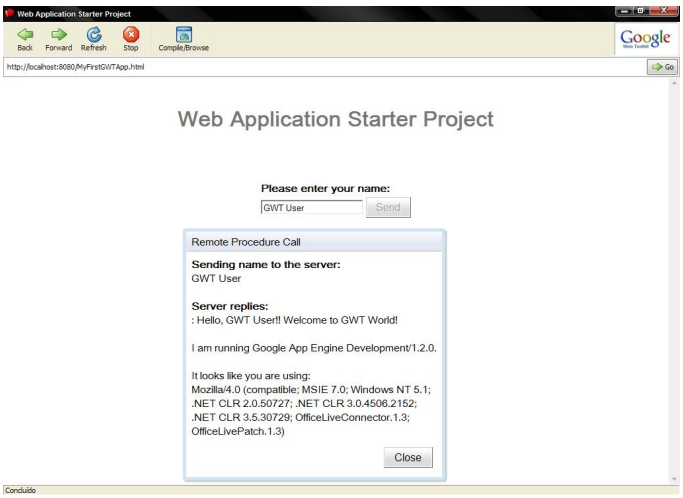


Figura 19 – Executando a aplicação (Hosted Mode)

Clicando em "Compile/Browse" poderá ver o resultado no browser. O que o GWT faz é compilar todo o código Java em Javascript (daí que o processo possa demorar um pouco) para este ser então executado no browser.

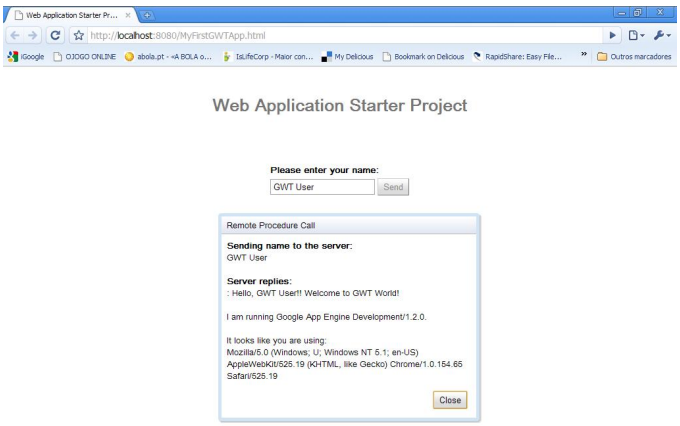


Figura 20 – Executando a aplicação (Web Mode)

Conclusão

Mais uma vez o Google Web Toolkit mostrou ser uma plataforma poderosa e em verdadeira ascensão, disponibilizando agora um novo plugin que vem automatizar e simplificar o uso da plataforma.

O anúncio de que o Google App Engine já suporta a linguagem Java foi uma grande novidade e, a sua integração no plugin da Google para o Eclipse, só veio motivar ainda mais o uso do GWT na medida em que os programadores conseguem ver publicadas as suas aplicações num piscar de olhos, não tendo de se preocupar com os aspectos que envolvem configuração do servidor.

O mecanismo RPC disponibilizado pelo GWT mostra-se muito útil e poderoso, pois vem facilitar o processo de comunicação cliente/servidor, flexibilizando o processo de troca de dados entre ambos.

São estas e muitas outras razões que tornam o Google Web Toolkit uma plataforma poderosa, agradável e atraente a quem desenvolve ou pretende desenvolver aplicações Web...

Espero que tenha achado o artigo útil. Até a próxima edição da Revista Programar!

Bibliografia

[Using the Google Plugin for Eclipse]
http://code.google.com/intl/pt-PT/appengine/docs/java/tools/eclipse.html
[Google Plugin for Eclipse]
http://www.infoq.com/br/news/2009/04/google-eclipse-plugin

[Remote Procedure Calls]
http://code.google.com/docreader/#p=google-web-toolkit-doc-1-5&s=google-web-toolkit-doc-1-5&t=DevGuideRemoteProcedureCalls

[GWT Tutorial - Building a GWT RPC Service]
http://developerlife.com/tutorials/?p=125

[Making Remote Procedure Calls]
http://code.google.com/intl/pt-PT/webtoolkit/tutorials/1.6/RPC.html#serialize

SOBRE O AUTOR

Licenciado em Engenharia de Informática pela Universidade da Madeira, está agora a concluir o Mestrado no respectivo curso na mesma universidade.Gosta de linguagens como Java e C#, mas no futuro profissional gostaria de enveredar pelo desenvolvimento de aplicações Web, dando foco a um tema ou aspecto que tem ganho cada vez mais relevância no Mundo do Software - a Usabilidade - neste caso concreto, a Usabilidade na Web.

cristian.goncalves@portugal-a-programar.org

Cristian Gonçalves

Arduino e a Aquisição de Dados

Introdução

É objectivo deste segundo artigo, abordando o Arduino, tentar explorar um problema que pode afectar os utilizadores que necessitem de uma maior velocidade de aquisição de dados para a sua aplicação, nomeadamente utilizando o Arduino.
Assim, abordaremos a problemática que envolve a taxa de amostragem, já que, para fazer a leitura de sensores (acelerómetros, termopares, entre outros) com esta "ferramenta", de uma forma eficiente, é necessário o controlo deste factor.
Se quisermos medir o comportamento de um objecto em queda durante 5 segundos, mas só "amostrarmos"- obter o valor da sua altura em relação ao solo, utilizando um sensor adequado, - de 15 em 15 segundos, facilmente se concluí que os dados obtidos não têm utilidade. Este exemplo ilustra assim a necessidade de abordar este problema, pois cada vez mais se está a utilizar o Arduino como "ferramenta" de aquisição de dados, especialmente devido à sua possibilidade de "interface" Ethernet ou Zigbee, através da sua capacidade de acrescentar um "shield".
O Arduino é uma ferramenta com enormes potencialidades, que cada vez mais começa a ser utilizada em projectos de relativa complexidade. Isto deve-se não só ao seu baixo custo, mas também à sua simplicidade de utilização.

Fig. 1 – Exemplo de "shield" Zigbee e Ethernet

Fundamento teórico

Na leitura de sensores, as amostras são geralmente adquiridas em intervalos fixos de tempo. Contudo, em casos específicos, a periodicidade de amostragem pode ser alterada.
Desta forma, quer aumentando ou diminuindo o intervalo de tempo entre amostras, a principal questão prende-se com a periodicidade da própria amostragem, visto que quanto maior for, maior terão de ser também a capacidade de processamento e de armazenamento.
O número de vezes em que se realiza a amostragem em uma unidade de tempo é a chamada taxa de amostragem, geralmente, medida em Hertz.
Dizer-se que a taxa de amostragem é de 100Hz, significa que em cada segundo são tomadas 100 medidas de uma determinada variação de voltagem do sinal de entrada. Dessa forma, quanto maior for a taxa de amostragem, mais exacta será a representação do sinal original.
Ao estudar-se a amostragem, é necessária a sensibilização para um dos fenómenos que podem ocorrer: o Aliasing.
Para o evitar deve-se respeitar o teorema Nyquist que, basicamente, diz-nos que a taxa de amostragem deve ser no mínimo duas vezes a maior frequência que desejamos registar. Se não respeitarmos este teorema, o sinal amostrado não poderá ser recuperado sem perda de informação.

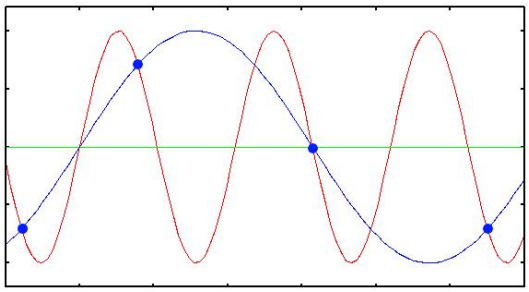


Fig. 2 – Exemplo de Aliasing

A onda azul na Figura 2 é a reconstrução da onda original (a vermelho), sem respeitar este teorema. E, como se pode constatar, não existe uma reconstrução da onda original, havendo assim perda de informação.
A solução para aumentar a taxa de amostragem no Arduino passa essencialmente por aumentar a velocidade do conversor analógico-digital (ADC) – como veremos de seguida.

ADC

O conversor analógico digital presente no ATmega168 (Arduino) possui 10 bit de resolução.
O ADC está preparado para um sinal de entrada analógica de tensão variável de 0V a 5V, gerando números binários de 0 (0000000000) a 1023 (1111111111), que dependem essencialmente do sinal colocado na entrada. (Ver Nuno Pessanha Santos, Introdução ao Arduino, Revista PROGRAMAR, 17º edição)
À excepção da primeira conversão que leva 25 ADC ciclos de clock, pois efectua a inicialização do ADC, uma conversão demora 13 ADC ciclos de clock. Ou seja, se tivermos um clock de 1 MHz vamos ter, aproximadamente:

$$Nº\ de\ amostras\ por\ segundo = \frac{10^6}{13} \approx 77000$$

O que vai limitar, como referido anteriormente, a largura de banda a utilizar para 38,5 kHz -segundo o teorema de Nyquist.

Arduino Vs ADC

Nem sempre a maior taxa de amostragem é a mais indicado. Seria errado pensarmos deste modo, pois ao aumentar a taxa de amostragem vamos, também, aumentar o espaço e o processamento necessário dos dados.
Assim, dever-se-á encontrar a menor taxa de amostragem que consiga satisfazer os nossos objectivos, pois não vamos ter quaisquer melhorias no nosso objectivo ao colocar taxas de amostragem excessivamente elevadas, já que poderemos estar a exigir esforços de processamento que não podemos fornecer.
O input clock do ADC encontra-se dividido por um "factor de divisão" – Prescaler. O valor deste factor pode ser modificado, alterando o valor do conteúdo do registo ADCSRA.

Bit	7	6	5	4	3	2	1	0
(0x7A)	ADEN	ADSC	ADATE	ADIF	ADIE	ADPS2	ADPS1	ADPS0
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Valor Inicial	0	0	0	0	0	0	0	0

Fig. 3 – Conteúdo do registo ADCSRA

ADPS2	ADPS1	ADPS0	Factor de divisão
0	0	0	2
0	0	1	2
0	1	0	4
0	1	1	8
1	0	0	16
1	0	1	32
1	1	0	64
1	1	1	128

Fig. 4 – Combinações possíveis

Os bits que nos interessam para este caso são o ADPS2, ADPS1 e o ADPS0.
Na tabela, podemos analisar quais as combinações destes três bits e a partir destas fazer alguns cálculos que nos possibilitam definir a taxa de amostragem adequada, por exemplo:
Se tivermos o bit 0 (ADPS0) a 1, o bit 1 (ADPS1) a 1 e o bit 2 (ADSP2) a 0, obtemos segundo a tabela acima um factor de divisão de 8.
O Arduino tem um clock de sistema de 16 MHz, obtendo assim um input clock para o ADC de 2 MHz.

$$Input\ clock\ ADC = \frac{16}{8} = 2\ MHz$$

O mesmo raciocínio seria tomado para qualquer factor de divisão possível - 2,4,8,16,32,64 e 128.
Destes 2 MHz obtidos para um factor de divisão de 8, temos de os dividir por 13 - Número de ciclos de clock gastos na conversão – para obter a respectiva taxa de amostragem.

$$Taxa\ de\ amostragem = \frac{2}{13} \approx 0{,}15\ MHz$$

Ou seja, com esta taxa obtemos 150000 amostras por cada segundo. Contudo, estes valores são apenas teóricos, não sendo estes os obtidos na realidade.

Como alterar o Prescaler

Exemplo de código fonte que permite alterar o valor do Prescaler.

```
(1)#ifndef cbi
(2)#define cbi(sfr, bit)
(_SFR_BYTE(sfr) &= ~_BV(bit))
(3)#endif
(4)#ifndef sbi
(5)#define sbi(sfr, bit)
(_SFR_BYTE(sfr) |= _BV(bit))
(6)#endif
(7)void setup(void) {
(8)Serial.begin(9600);
(9)sbi(ADCSRA, ADPS2);
(10)cbi(ADCSRA, ADPS1);
(11)cbi(ADCSRA, ADPS0);
(12)}
(13)int a;
(14)int b;
(15)int c;
(16)void loop()
(17){ b = micros();
(18)a = analogRead(0);
(19)c = micros();
(20)Serial.print(c-b);
(21)Serial.print('-');
(22)Serial.println(a);
(23)}
```

No exemplo de código acima apresentado, é possível fazer uma leitura de quanto tempo é gasto, na realidade, a efectuar a conversão de analógico para digital.
Nas linhas 9,10 e 11 é estabelecido o conteúdo dos registos responsáveis pelo factor de divisão. A instrução sbi atribui o valor 1 e a instrução cbi atribui o valor 0, sendo a instrução constituída por:

Instrução (registo, bit a estabelecer);

Analisando a instrução da linha 9, a título de exemplo, podemos ver que é utilizada a instrução sbi (Atribui o valor 1)

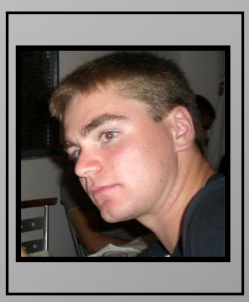
ao bit 2 (ADPS2) do registo ADCSRA.
Através da instrução Serial.begin, são enviados, pela porta série, os valores do tempo de conversão. (Respectivamente c-b)
Deste modo, ao fazer a diferença entre o valor c (linha 20) e o valor b (linha 18), vamos obter o valor do tempo dispendido a executar a função analogRead. Isto é, obtemos assim o valor real de tempo dispendido a efectuar a conversão de analógico para digital.
Não obtendo, como foi referido anteriormente, os valores teóricos calculados.

Conclusão

O tema aqui abordado permite uma melhoria das capacidades do Arduino como datalogger.

É fácil perceber que, recorrendo ao "datasheet" do microcontrolador e sabendo o nome dos registos existentes, é possível uma configuração "mais personalizada" desta "ferramenta", isto é, qualquer registo existente é assim passível de ser configurado. O que abre "portas" para um rendimento mais elevado, nas aplicações desejadas.
Este artigo não representa, obviamente, tudo o que há a aprender sobre este tema, tendo o leitor de, caso tenha necessidade, aprofundar os seus conhecimentos nesta área.
Se é o primeiro contacto com o Arduino ou sentiu curiosidade em aprender, leia o artigo "Introdução ao Arduino" na 17º edição da revista PROGRAMAR.
Pois o saber não ocupa espaço de memória.

SOBRE O AUTOR



Nuno Pessanha Santos é um apaixonado pela área da electrónica e das telecomunicações, estando actualmente a frequentar na Escola Naval o primeiro ano do mestrado integrado no ramo de Armas e Electrónica na classe de Engenheiros Navais.

nuno.santos@portugal-a-programar.org

Nuno Santos

Queres participar na Revista PROGRAMAR? Queres integrar este projecto, escrever artigos e ajudar a tornar esta revista num marco da programação nacional?

Vai a

www.revista-programar.info

para mais informações em como participar,
ou então contacta-nos por

revistaprogramar
@portugal-a-programar.org

Precisamos do apoio de todos para tornar este projecto ainda maior...

contamos com a tua ajuda!

Equipa PROGRAMAR

Um projecto Portugal-a-Programar.org