Stockez n'importe quel type avec Boost.Any

par Alp Mestan (Site perso de Alp) (Blog)

Date de publication :

Dernière mise à jour :

Cet article présente le module **Any** de la bibliothèque dont la réputation n'est plus à faire : **Boost** .

Copyright © 2008 - Alp Mestan. Aucune reproduction, même partielle, ne peut être faite de ce site et de l'ensemble de son contenu : textes, documents, images, etc sans l'autorisation expresse de l'auteur. Sinon vous encourez selon la loi jusqu'à 3 ans de prison et jusqu'à 300 000 E de domages et intérêts.

I - Introduction

Le module Boost.Any permet de stocker une variable de n'importe quel type dans un objet de type *boost::any* , bien qu'il y ait des pré-requis sur le type en question, ce que nous verrons plus loin dans l'article.

Vous pouvez consulter la documentation officielle de Boost.Any à l'adresse suivante : **Boost.Any**

Copyright © 2008 - Alp Mestan. Aucune reproduction, même partielle, ne peut être faite de ce site et de l'ensemble de son contenu : textes, documents, images, etc sans l'autorisation expresse de l'auteur. Sinon vous encourez selon la loi jusqu'à 3 ans de prison et jusqu'à 300 000 E de domages et intérêts.

II - Présentation du module Boost.Any

Comme vous pouvez le voir sur la documentation officielle de Boost.Any, la bibliothèque se "résume" à ceci.

Elements clés du module Boost.Any
```
namespace boost {
  class bad_any_cast;
  class any;
  template<typename T> T any_cast(any &);
  template<typename T> T any_cast(const any &);
  template<typename ValueType> const ValueType * any_cast(const any *);
  template<typename ValueType> ValueType * any_cast(any *);
}
```

II-1 - La classe boost::any

Nous n'étudierons pas ici le fonctionnement interne de la classe *boost::any* mais allons plutôt regarder l'interface de cette dernière. Si toutefois vous désirez voir comment la classe *boost::any* fonctionne, vous pouvez regarder **son code** .

Voici l'interface de la classe *boost::any* .

```
class any
{
 public :
 any();
 template <typename ValueType> any(const ValueType & value);
 any(const any & other);

 ~any();

 any & swap(any & rhs);
 template <typename ValueType> any & operator=(const ValueType & rhs);
 any & operator=(const any & rhs);
 bool empty() const;
 const std::type_info & type() const;

 // nous n'étudions pas le reste de la classe
};
```

Etudions maintenant chacun des éléments de l'interface de *boost::any* .

Les constructeurs :

- *any()* : initialise l'objet contenu (qui est un pointeur) à 0 ;
- *template <typename ValueType> any(const ValueType & value)* : mets *value* dans l'objet contenu dans l'instance de *boost::any* ;
- *any(const any & other)* : place dans l'instance courante l'objet contenu dans *other* ;

Le destructeur : détruit l'objet contenu dans l'instance courante, s'il n'est pas déjà vide (0) ;

Les fonctions membres et opérateurs publics :

- *any & swap(any & rhs)* : échange le contenu de l'instance courante avec celui de *rhs* ;

Copyright © 2008 - Alp Mestan. Aucune reproduction, même partielle, ne peut être faite de ce site et de l'ensemble de son contenu : textes, documents, images, etc sans l'autorisation expresse de l'auteur. Sinon vous encourez selon la loi jusqu'à 3 ans de prison et jusqu'à 300 000 E de domages et intérêts.
http://alp.developpez.com/

- *template <typename ValueType> any & operator=(const ValueType & rhs)* : remplace l'objet contenu par l'instance courante par l'objet *rhs* ;
- *any & operator=(const any & rhs)* : remplace l'objet contenu par l'instance courante par l'objet contenu par l'instance *rhs* ;
- *bool empty() const* : retourne vrai si aucun objet n'est contenu par l'instance courante, faux sinon ;
- *const std::type_info & type() const* : retourne le *std::type_info* associé au type réel de l'objet contenu dans l'instance courante.

II-2 - La classe bad_any_cast

La classe *boost::bad_any_cast* représente l'exception qui est levée par *boost::any_cast* (que nous verrons plus loin) en cas d'échec. Voici ce à quoi elle se résume :

```
class bad_any_cast : public std::bad_cast {
public:
  virtual const char * what() const;
};
```

Voyons maintenant comment convertir un *boost::any* en un type précis, lorsque c'est possible.

II-3 - Les fonctions de conversion

Voici le prototype des fonctions de conversion.

```
template <typename T> T any_cast(any &);
template <typename T> T any_cast(const any &);
template <typename ValueType> const ValueType * any_cast(const any *);
template <typename ValueType> ValueType * any_cast(any *);
```

Si l'on passe un pointeur à un *boost::any_cast* , ce dernier retourne un pointeur vers l'objet contenu si la conversion est réussie, un pointeur nul sinon. Si T est un type à sémantique de valeur, *boost::any_cast* retourne une copie de l'objet contenu. Si T est une référence sur un type à sémantique de valeur, *boost::any_cast* retourne une référence sur l'objet stocké. Tout cela bien sûr sous réserve que le type demandé et le type réel de l'objet contenu soient compatibles.

II-4 - Les contraintes sur le type qui est contenu

Les contraintes sur le type *ValueType* de l'objet que va stocker un *boost::any* sont celles-ci :

- *ValueType* doit être copiable ;
- *ValueType* doit être assignable ; toutes les formes d'assignation utilisables avec *ValueType* doivent être **exception-safe** ;
- Le destructeur (si *ValueType* est une classe) ne doit lancer aucune exception, quoiqu'il arrive.

Copyright © 2008 - Alp Mestan. Aucune reproduction, même partielle, ne peut être faite de ce site et de l'ensemble de son contenu : textes, documents, images, etc sans l'autorisation expresse de l'auteur. Sinon vous encourez selon la loi jusqu'à 3 ans de prison et jusqu'à 300 000 E de domages et intérêts.

III - Exemples d'utilisation

Nous allons maintenant essayer de nous familiariser avec l'utilisation du module **Boost.Any** par le biais de quelques exemples.

Tout d'abord, sachez qu'il y a un en-tête de boost à inclure pour utiliser la classe *boost::any* , et que cette dernière est dans l'espace de nom de boost.

Instructions indispensables pour l'utilisation de Boost.Any

```
#include <boost/any.hpp>

using namespace boost;
```

III-1 - Utilisation de base

```
#include <iostream>
#include <boost/any.hpp>

using namespace boost;
using namespace std;

int main()
{

 any a1(5); // création d'un boost::any contenant un entier
 cout << any_cast<int>(a1) << endl; // récupération puis affichage de l'entier 5
 any a2('c'); // création d'un boost::any contenant un caractère
 cout << any_cast<char>(a2) << endl; // récupération puis affichage du caractère 'c'
 a1 = a2; // appel à l'opérateur = de la classe boost::any : on remplace 5 par 'c' dans a1
 cout << any_cast<char>(a1) << endl; // affiche le caractère 'c'

 return 0;
}
```

Ce code met en oeuvre les bases de l'utilisation de boost::any et des fonctions any_cast. Voyons maintenant le système d'exception.

III-2 - Mise en oeuvre de bad_any_cast

Le module **Boost.Any** fournit une classe représentant une exception, lancée en cas d'échec de conversion. Vous pourrez la voir en oeuvre avec l'exemple suivant.

```
#include <iostream>
#include <boost/any.hpp>

using namespace boost;
using namespace std;

// définition d'une structure, pour les besoins de l'exemple
struct S
{
};

int main()
{
 S s;
 any a(s); // on stocke l'instance de S dans notre boost::any
```


Copyright © 2008 - Alp Mestan. Aucune reproduction, même partielle, ne peut être faite de ce site et de l'ensemble de son contenu : textes, documents, images, etc sans l'autorisation expresse de l'auteur. Sinon vous encourez selon la loi jusqu'à 3 ans de prison et jusqu'à 300 000 E de domages et intérêts.

```
 try
 {
  char c = any_cast<char>(a); // on essaye de convertir le boost::any contenant une instance de S
 en un caractère
  // cela va échouer et lancer une exception de type bad_any_cast
 }

 catch(bad_any_cast& bac) // nous attrapons l'exception
 {
  cout << bac.what() << endl; // on affiche le message contenu dans l'exception
 }

 return 0;
}
```

Les deux exemples précédents sont assez simples et soulignent les fonctionnalités clés de ce module. Nous allons maintenant regarder un dernier exemple, un peu plus compliqué, qui sera surement plus parlant.

III-3 - Exemple plus complexe

Voyons maintenant ce que peut donner le couplage de la classe *boost::any* à un conteneur standard, comme *std::vector* .

```
#include <iostream>
#include <vector>
#include <algorithm>
#include <boost/any.hpp>

using namespace boost;
using namespace std;

// définition d'une classe, pour l'exemple
class MaClasse
{
 std::string s;

 public :
  MaClasse(const std::string& s_) : s(s_)
  { }
};

// 4 fonctions permettant d'établir une correspondance entre le type réel de l'objet stocké
// et les types char, int, std::string et MaClasse
bool is_char(const any& a)
{
 return a.type() == typeid(char);
}

bool is_int(const any& a)
{
 return a.type() == typeid(int);
}

bool is_std_string(const any& a)
{
 return any_cast<std::string>(&a);
}

bool is_ma_classe(const any& a)
{
 return a.type() == typeid(MaClasse);
}

int main()
```


Copyright © 2008 - Alp Mestan. Aucune reproduction, même partielle, ne peut être faite de ce site et de l'ensemble de son contenu : textes, documents, images, etc sans l'autorisation expresse de l'auteur. Sinon vous encourez selon la loi jusqu'à 3 ans de prison et jusqu'à 300 000 E de domages et intérêts.

```
{
 // création de nos variables pour le std::vector
 int i1 = 2;
 int i2 = 45;
 int i3 = 2321;
 char c1 = 'd';
 char c2 = 'v';
 char c3 = 'p';
 std::string s1 = "Developpez";
 std::string s2 = ".com";
 MaClasse mc1("Boost.Any");
 MaClasse mc2("permet de stocker n'importe quel type");

 // création d'un vecteur contenant des boost::any, donc pouvant contenir presque
 // n'importe quoi
 vector<any> values;

 // ajout des variables dans le vecteur
 values.push_back(i1);
 values.push_back(i2);
 values.push_back(i3);
 values.push_back(c1);
 values.push_back(c2);
 values.push_back(c3);
 values.push_back(s1);
 values.push_back(s2);
 values.push_back(mc1);
 values.push_back(mc2);

 // on compte le nombre d'entiers, de caracteres, de std::string et de MaClasse
 // présents dans le vecteur
 cout << "Nombre d'entiers : " << count_if(values.begin(), values.end(), is_int) << endl;
 cout << "Nombre de caracteres : " << count_if(values.begin(), values.end(), is_char) << endl;
 cout << "Nombre de std::string : " << count_if(values.begin(), values.end(), is_std_string) <<
 endl;
 cout << "Nombre de MaClasse : " << count_if(values.begin(), values.end(), is_ma_classe) << endl;

 return 0;
}
```

Nous pouvons remarquer au passage qu'un *std::vector<boost::any>* permet de stocker des données hétérogènes pouvant être d'un type presque quelconque.

Copyright © 2008 - Alp Mestan. Aucune reproduction, même partielle, ne peut être faite de ce site et de l'ensemble de son contenu : textes, documents, images, etc sans l'autorisation expresse de l'auteur. Sinon vous encourez selon la loi jusqu'à 3 ans de prison et jusqu'à 300 000 E de domages et intérêts.

IV - Conclusion

Cet article a donc introduit un petit mais non moindre module de la bibliothèque Boost, le module **Boost.Any** . Si vous avez des questions, n'hésitez pas à vous rendre **sur le forum Boost de Developpez** , en mettant en début de titre le tag *[Any]* .

Copyright © 2008 - Alp Mestan. Aucune reproduction, même partielle, ne peut être faite de ce site et de l'ensemble de son contenu : textes, documents, images, etc sans l'autorisation expresse de l'auteur. Sinon vous encourez selon la loi jusqu'à 3 ans de prison et jusqu'à 300 000 E de domages et intérêts.

V - Remerciements

Merci à ... pour la relecture de cet article.

Copyright © 2008 - Alp Mestan. Aucune reproduction, même partielle, ne peut être faite de ce site et de l'ensemble de son contenu : textes, documents, images, etc sans l'autorisation expresse de l'auteur. Sinon vous encourez selon la loi jusqu'à 3 ans de prison et jusqu'à 300 000 E de domages et intérêts.

Copyright © 2008 - Alp Mestan. Aucune reproduction, même partielle, ne peut être faite de ce site et de l'ensemble de son contenu : textes, documents, images, etc sans l'autorisation expresse de l'auteur. Sinon vous encourez selon la loi jusqu'à 3 ans de prison et jusqu'à 300 000 E de domages et intérêts.