G++ 2.91.57，cygnus\cygwin-b20\include\g++\type_traits.h 完整列表

```
/*
 *
 * Copyright (c) 1997
 * Silicon Graphics Computer Systems, Inc.
 *
 * Permission to use, copy, modify, distribute and sell this software
 * and its documentation for any purpose is hereby granted without fee,
 * provided that the above copyright notice appear in all copies and
 * that both that copyright notice and this permission notice appear
 * in supporting documentation.  Silicon Graphics makes no
 * representations about the suitability of this software for any
 * purpose.  It is provided "as is" without express or implied warranty.
 */

#ifndef __TYPE_TRAITS_H
#define __TYPE_TRAITS_H

#ifndef __STL_CONFIG_H
#include <stl_config.h>
#endif

/*
本檔提供了一個框架（framework），允許針對型別屬性（type attributes），在編譯時期完
成函式派送（dispatch）。這對於撰寫 template 很有幫助。例如，當我們準備對一個「元
素型別未知」的陣列執行拷貝（copy）動作時，如果我們能事先知道其元素型別是否有一個
trivial copy constructor，便能夠幫助我們決定是否可使用快速的 memcpy()。

class template __type_traits 提供許多typedefs，每一個若非 __true_type
就是 __false_type。__type_traits 的引數可以是任何型別。這些typedefs將經
由以下管道獲得正確值：
    1. 一般具現體（general instantiation），內含對所有型別而言都必定有效的保守值。
    2. 經過宣告的特化版本（例如本檔對所有 C++ 內建型別所提供的特化宣告）。
    3. 某些編譯器（如Silicon Graphics N32 和 N64 編譯器）會自動為所有型別提供
       適當的特化版本。

舉例：

// 拷貝一個陣列，其元素型別擁有 non-trivial copy constructors。
template <class T> void copy(T* source,T* destination,int n,__false_type);

// 拷貝一個陣列，其元素型別擁有 trivial copy constructors。運用 memcpy() 完成工作。
template <class T> void copy(T* source,T* destination,int n,__true_type);

// 拷貝一個陣列，其元素為任意型別，視情況採用最有效率的拷貝機制。
template <class T> inline void copy(T* source,T* destination,int n)
{
  copy(source,destination,n,
       typename __type_traits<T>::has_trivial_copy_constructor());
```

```
}
*/


struct __true_type {
};

struct __false_type {
};

template <class type>
struct __type_traits {
   typedef __true_type     this_dummy_member_must_be_first;
           /* 不要移除這個成員。它通知「有能力自動將 __type_traits 特化」
           的編譯器說，我們現在所看到的這個 __type_traits template 是特
           殊的。這是為了確保萬一編譯器也使用一個名為 __type_traits 而其
           實與此處定義並無任何關聯的 template 時，所有事情都仍將順利運作。
           */

   /* 以下條件應被遵守，因為編譯器有可能自動為各型別產生專屬的 __type_traits
      特化版本：
          - 你可以重新排列以下的成員次序
          - 你可以移除以下任何成員
          - 絕對不可以將以下成員重新命名而卻沒有改變編譯器中的對應名稱
          - 新加入的成員會被視為一般成員，除非你在編譯器中加上適當支援。*/

   typedef __false_type    has_trivial_default_constructor;
   typedef __false_type    has_trivial_copy_constructor;
   typedef __false_type    has_trivial_assignment_operator;
   typedef __false_type    has_trivial_destructor;
   typedef __false_type    is_POD_type;
   // 所謂 POD 意指 Plain Old Data structure.
};

// 以下提供某些特化版本。這對於內建有 __types_traits 支援能力的編譯器並無
// 傷害，而對於無該等支援能力的編譯器而言則屬必要。

/* 以下針對 C++ 基本型別 char, signed char, unsigned char, short, unsigned
short, int, unsigned int, long, unsigned long, float, double, long
double 提供特化版本。注意，每一個成員的值都是 __true_type，表示這些型別都可
採用最快速方式（例如 memcpy）來進行拷貝動作或賦值動作。*/

// 當編譯器支援 partial specialization，__STL_TEMPLATE_NULL 被定義為
// template<>，見 <stl_config.h>

__STL_TEMPLATE_NULL struct __type_traits<char> {
   typedef __true_type    has_trivial_default_constructor;
   typedef __true_type    has_trivial_copy_constructor;
   typedef __true_type    has_trivial_assignment_operator;
```

```
    typedef __true_type    has_trivial_destructor;
    typedef __true_type    is_POD_type;
};

__STL_TEMPLATE_NULL struct __type_traits<signed char> {
    typedef __true_type    has_trivial_default_constructor;
    typedef __true_type    has_trivial_copy_constructor;
    typedef __true_type    has_trivial_assignment_operator;
    typedef __true_type    has_trivial_destructor;
    typedef __true_type    is_POD_type;
};

__STL_TEMPLATE_NULL struct __type_traits<unsigned char> {
    typedef __true_type    has_trivial_default_constructor;
    typedef __true_type    has_trivial_copy_constructor;
    typedef __true_type    has_trivial_assignment_operator;
    typedef __true_type    has_trivial_destructor;
    typedef __true_type    is_POD_type;
};

__STL_TEMPLATE_NULL struct __type_traits<short> {
    typedef __true_type    has_trivial_default_constructor;
    typedef __true_type    has_trivial_copy_constructor;
    typedef __true_type    has_trivial_assignment_operator;
    typedef __true_type    has_trivial_destructor;
    typedef __true_type    is_POD_type;
};

__STL_TEMPLATE_NULL struct __type_traits<unsigned short> {
    typedef __true_type    has_trivial_default_constructor;
    typedef __true_type    has_trivial_copy_constructor;
    typedef __true_type    has_trivial_assignment_operator;
    typedef __true_type    has_trivial_destructor;
    typedef __true_type    is_POD_type;
};

__STL_TEMPLATE_NULL struct __type_traits<int> {
    typedef __true_type    has_trivial_default_constructor;
    typedef __true_type    has_trivial_copy_constructor;
    typedef __true_type    has_trivial_assignment_operator;
    typedef __true_type    has_trivial_destructor;
    typedef __true_type    is_POD_type;
};

__STL_TEMPLATE_NULL struct __type_traits<unsigned int> {
    typedef __true_type    has_trivial_default_constructor;
    typedef __true_type    has_trivial_copy_constructor;
    typedef __true_type    has_trivial_assignment_operator;
    typedef __true_type    has_trivial_destructor;
```

```
  typedef __true_type    is_POD_type;
};

__STL_TEMPLATE_NULL struct __type_traits<long> {
  typedef __true_type    has_trivial_default_constructor;
  typedef __true_type    has_trivial_copy_constructor;
  typedef __true_type    has_trivial_assignment_operator;
  typedef __true_type    has_trivial_destructor;
  typedef __true_type    is_POD_type;
};

__STL_TEMPLATE_NULL struct __type_traits<unsigned long> {
  typedef __true_type    has_trivial_default_constructor;
  typedef __true_type    has_trivial_copy_constructor;
  typedef __true_type    has_trivial_assignment_operator;
  typedef __true_type    has_trivial_destructor;
  typedef __true_type    is_POD_type;
};

__STL_TEMPLATE_NULL struct __type_traits<float> {
  typedef __true_type    has_trivial_default_constructor;
  typedef __true_type    has_trivial_copy_constructor;
  typedef __true_type    has_trivial_assignment_operator;
  typedef __true_type    has_trivial_destructor;
  typedef __true_type    is_POD_type;
};

__STL_TEMPLATE_NULL struct __type_traits<double> {
  typedef __true_type    has_trivial_default_constructor;
  typedef __true_type    has_trivial_copy_constructor;
  typedef __true_type    has_trivial_assignment_operator;
  typedef __true_type    has_trivial_destructor;
  typedef __true_type    is_POD_type;
};

__STL_TEMPLATE_NULL struct __type_traits<long double> {
  typedef __true_type    has_trivial_default_constructor;
  typedef __true_type    has_trivial_copy_constructor;
  typedef __true_type    has_trivial_assignment_operator;
  typedef __true_type    has_trivial_destructor;
  typedef __true_type    is_POD_type;
};

#ifdef __STL_CLASS_PARTIAL_SPECIALIZATION

template <class T>
struct __type_traits<T*> {
  typedef __true_type    has_trivial_default_constructor;
  typedef __true_type    has_trivial_copy_constructor;
```

```
   typedef __true_type    has_trivial_assignment_operator;
   typedef __true_type    has_trivial_destructor;
   typedef __true_type    is_POD_type;
};

#else /* __STL_CLASS_PARTIAL_SPECIALIZATION */

struct __type_traits<char*> {
   typedef __true_type    has_trivial_default_constructor;
   typedef __true_type    has_trivial_copy_constructor;
   typedef __true_type    has_trivial_assignment_operator;
   typedef __true_type    has_trivial_destructor;
   typedef __true_type    is_POD_type;
};

struct __type_traits<signed char*> {
   typedef __true_type    has_trivial_default_constructor;
   typedef __true_type    has_trivial_copy_constructor;
   typedef __true_type    has_trivial_assignment_operator;
   typedef __true_type    has_trivial_destructor;
   typedef __true_type    is_POD_type;
};

struct __type_traits<unsigned char*> {
   typedef __true_type    has_trivial_default_constructor;
   typedef __true_type    has_trivial_copy_constructor;
   typedef __true_type    has_trivial_assignment_operator;
   typedef __true_type    has_trivial_destructor;
   typedef __true_type    is_POD_type;
};

#endif /* __STL_CLASS_PARTIAL_SPECIALIZATION */


#endif /* __TYPE_TRAITS_H */

// Local Variables:
// mode:C++
// End:
```