2017 CPP-Summit

C++ Concept 的前世今生

吴咏炜

wuyongwei@gmail.com

一个例子

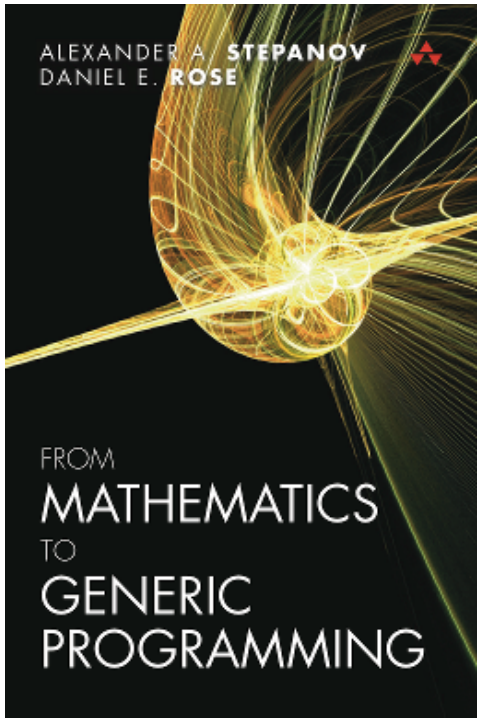

```
template <typename N>
bool odd(N n)
{
    return bool(n & 0x1);
}

std::cout << odd(1) << std::endl;
std::cout << odd(1.0) << std::endl; // Does not compile
std::cout << odd('a') << std::endl; // This compiles!
```

一个例子


```
template <typename N>
bool odd(N n)
{
    return bool(n & 0x1);
}
```

```
template <Integer N>
bool odd(N n)
{
    return bool(n & 0x1);
}
```



等价形式


```
inline bool odd(Integer n)
{
    return bool(n & 0x1);
}
```

```
template <typename N>
inline bool odd(N n)
    requires(Integer<N>)
{
    return bool(n & 0x1);
}
```

为什么需要 Concept ?

- To make requirement on types used as template arguments explicit
  - Precise documentation
  - Better error messages
  - Overloading

C++11 之前的标准化努力

- Bjarne Stroustrup and Gabriel Dos Reis: Concepts — Design choices for template argument checking. October 2003. An early discussion of design criteria for "concepts" for C++.
- Bjarne Stroustrup: Concept checking — A more abstract complement to type checking. October 2003. A discussion of models of "concept" checking.
- Bjarne Stroustrup and Gabriel Dos Reis: A concept design (Rev. 1). April 2005. An attempt to synthesize a "concept" design based on (among other sources) N1510, N1522, and N1536.
- Jeremy Siek, Douglas Gregor, Ronald Garcia, Jeremiah Willcock, Jaakko Jarvi, and Andrew Lumsdaine: Concepts for C++0x. N1758==05-0018. May 2005.
- Gabriel Dos Reis and Bjarne Stroustrup: Specifying C++ Concepts. POPL06. January 2006.
- D. Gregor, B. Stroustrup: Concepts. N2042==06-0012. June 2006. The basis for all further "concepts" work for C++0x.
- Douglas Gregor, Jaakko Jarvi, Jeremy Siek, Bjarne Stroustrup, Gabriel Dos Reis, Andrew Lumsdaine: Concepts: Linguistic Support for Generic Programming in C++. OOPSLA'06, October 2006. An academic paper on the C++0x design and its experimental compiler "ConceptGCC."
- Pre-Frankfurt working paper (with "concepts" in the language and standard library): http://www.open-std.org/jtc1/sc22/wg21/docs/papers/2009/n2914.pdf . N2914=09-0104. June 2009.
- B. Stroustrup: Simplifying the use of concepts. N2906=09-0096. June 2009.

Explicit Concept Map


```
concept LessThanComparable<typename T> {
    bool operator<(const T& a, const T& b);
    bool operator>(const T& a, const T& b) { return b < a; }
    bool operator<=(const T& a, const T& b) { return !(b < a); }
    bool operator>=(const T& a, const T& b) { return !(a < b); }
}

concept_map LessThanComparable<int> {}

struct name { char* first; char* last; };

int namecmp(const name& n1, const name& n2)
{
    int c = strcmp(n1.last, n2.last);
    if (c == 0) return strcmp(n1.first, n2.first);
    else return c;
}

concept_map LessThanComparable<name> {
    bool operator<(const name& a, const name& b)
    {
        return namecmp(a, b) < 0;
    }
}
```

概念没有进入 C++11 的原因

- 草案复杂
  - Concepts
  - Concept maps
  - Axioms
- 争议颇多
- 没有成熟实现

Bjarne 对于概念的原则（2009）

- 鸭子类型
  - 泛型编程模板成功的关键
  - 不像面向对象一样需要明确使用接口
- 替换性
  - 不会匹配/调用比“保证”的先决条件要求更高的函数
- “意外匹配”是个小问题
  - 不会是 C++ 的前 100 个问题

2017 CPP-Summit

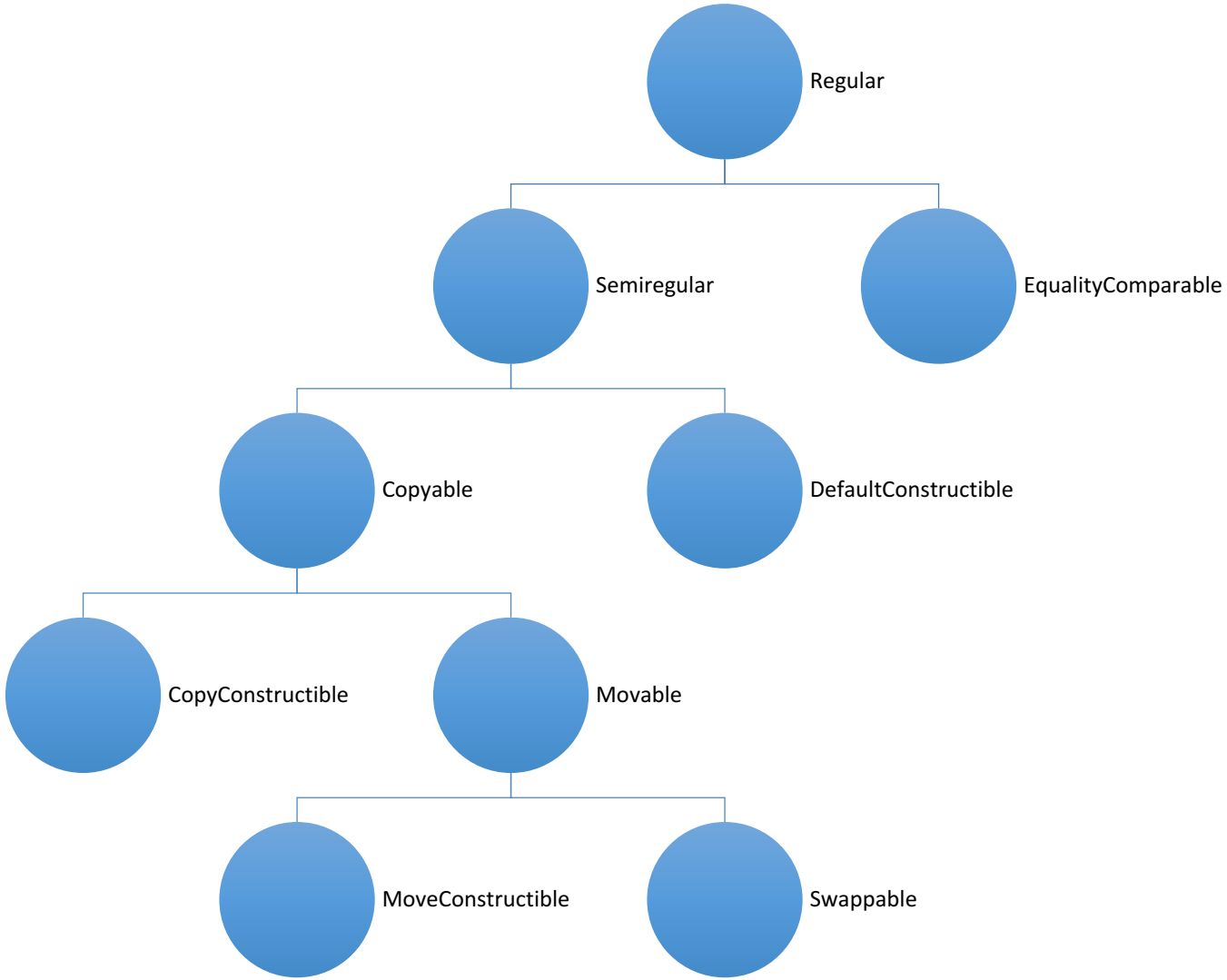
Concept TS

(2015/11)

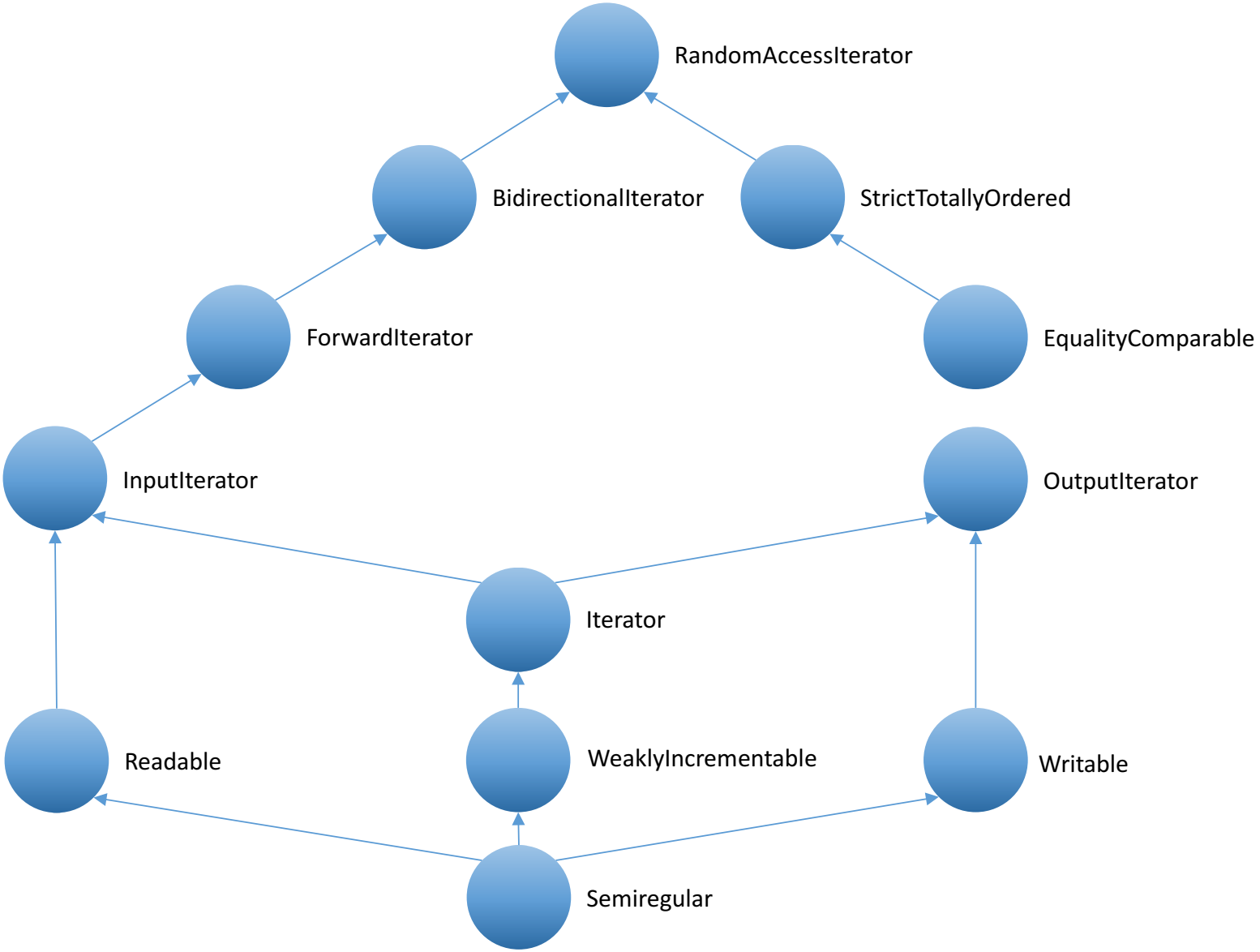
Concept 没有进入 C++17 的原因

- 从 Concept TS 出版到标准定稿只有不到四个月的时间
- 概念只有一个实现（GCC）
- Concept TS 规格书的作者和 GCC 中的概念实现者是同一个人；没有人独立地从规格书出发实现概念
- 目前只有 Ranges TS 及其实现 cmcstl2 大量使用了概念
- Concept TS 里没有实际定义概念（Ranges TS 里有）

Ranges TS 里的对象常用概念（部分）



Ranges TS 里的迭代器相关概念（部分）



CMCSTL2 里的部分实际概念定义 I


```
template <class T, class U>
concept bool WeaklyEqualityComparable =
    requires(const remove_reference_t<T>& t,
             const remove_reference_t<U>& u) {
        { t == u } -> Boolean&&;
        { t != u } -> Boolean&&;
        { u == t } -> Boolean&&;
        { u != t } -> Boolean&&;
    };

template <class T>
concept bool EqualityComparable =
    WeaklyEqualityComparable<T, T>;
```

CMCSTL2 里的部分实际概念定义 II


```
template <class T>
concept bool Copyable =
    CopyConstructible<T> &&
    Movable<T> &&
    Assignable<T&, const T&>;

template <class T>
concept bool Semiregular =
    Copyable<T> && DefaultConstructible<T>;

template <class T>
concept bool Regular =
    Semiregular<T> && EqualityComparable<T>;
```

CMCSTL2 里的部分实际概念定义 III


```
template <class I>
concept bool WeaklyIncrementable =
    Semiregular<I> &&
    requires(I& i) {
        typename difference_type_t<I>;
        { ++i } -> Same<I&>&&;
        i++;
    };

template <class I>
concept bool Iterator =
    detail::Dereferenceable<I&> && WeaklyIncrementable<I>;

template <class I, class T>
concept bool OutputIterator =
    Iterator<I> &&
    Writable<I, T> &&
    requires(I& i, T&& t) {
        *i++ = std::forward<T>(t);
    };
```

Concept 测试


```
#define TEST_CONCEPT(Concept, Type...) \
    cout << #Concept << '<' << #Type << ">: " << Concept<Type> << endl

TEST_CONCEPT(Semiregular, std::unique_ptr<int>);
TEST_CONCEPT(EqualityComparable, std::unique_ptr<int>);
TEST_CONCEPT(Readable, int);
TEST_CONCEPT(Readable, std::unique_ptr<int>);
TEST_CONCEPT(Writable, std::unique_ptr<int>, int);
TEST_CONCEPT(Semiregular, arma::imat);
TEST_CONCEPT(Assignable, arma::imat&, arma::imat&);
TEST_CONCEPT(Semiregular, arma::imat22);
TEST_CONCEPT(Assignable, arma::imat22&, arma::imat22&);
TEST_CONCEPT(Regular, int);
TEST_CONCEPT(Regular, char);
TEST_CONCEPT(Integer, int);
TEST_CONCEPT(Integer, char);
```

Concept 测试结果


```
Semiregular<std::unique_ptr<int>>: 0
EqualityComparable<std::unique_ptr<int>>: 1
Readable<int>: 0
Readable<std::unique_ptr<int>>: 1
Writable<std::unique_ptr<int>, int>: 1
Semiregular<arma::imat>: 1
Assignable<arma::imat&, arma::imat&>: 1
Semiregular<arma::imat22>: 0
Assignable<arma::imat22&, arma::imat22&>: 0
Regular<int>: 1
Regular<char>: 1
Integer<int>: 1
Integer<char>: 0
```

Sutton 的例子


```
template <typename R, typename T>
bool in(R const& range, T const& value)
    requires Range<R> &&
             EqualityComparableWith<T, value_type_t<R>>
{
    for (auto const& x : range)
        if (x == value)
            return true;
    return false;
}

vector<string> v{"Hello", "World"};
in(v, 0);
```

出错信息

- GCC 7.2 缺省输出
  - 166 行
- Apple Clang 缺省输出
  - 80 行
- GCC 7.2 -std=c++17 -fconcepts 输出

```
sutton_example.cpp: In function 'int main()':
sutton_example.cpp:26:12: error: cannot call function 'bool in(const R&, const
T&) requires  Range<R> and EqualityComparableWith<T, typename
std::experimental::ranges::v1::value_type<T>::type> [with R =
std::vector<std::__cxx11::basic_string<char> >; T = int]'
     in(v, 0);
           ^
sutton_example.cpp:13:6: note:   constraints not satisfied
 bool in(R const& range, T const& value)
      ^~
sutton_example.cpp:13:6: note: in the expansion of concept
'EqualityComparableWith<T, typename
std::experimental::ranges::v1::value_type<T>::type>' template<class T, class
U> concept const bool std::experimental::ranges::v1::EqualityComparableWith<T,
U> [with T = int; U =
std::experimental::ranges::v1::value_type<std::vector<std::__cxx11::basic_stri
ng<char> > >::type]
```

概念和重载


```
template <typename R>
bool in(R const& range, size_t value)
    requires requires(R r) { r.size(); }
{
    if (range.size() == value)
        return true;
    return false;
}

vector<string> v{"Hello", "World"};
in(v, "Hello");
in(v, 0);
```

不使用 Concept 时的变通重载方法


```
template <typename _Fn, typename _T1, typename _T2>
constexpr auto fmap(_Fn&& f, const std::pair<_T1, _T2>& args);

template <typename _Fn, typename... _Targs>
constexpr auto fmap(_Fn&& f, const std::tuple<_Targs...>& args);

template <template <typename, typename> class _OutCont = std::vector,
          template <typename> class _Alloc = std::allocator,
          typename _Fn, class _Rng>
constexpr auto fmap(_Fn&& f, _Rng&& inputs) -> decltype(
    begin(inputs), end(inputs),
    _OutCont<
        std::decay_t<decltype(f(*begin(inputs)))>,
        _Alloc<std::decay_t<decltype(f(*begin(inputs)))>>>());
```

另一个 SFINAE


```
template <class _T1, class _T2>
struct can_reserve {
    struct good { char dummy; };
    struct bad { char dummy[2]; };
    template <class _Up, void (_Up::*)(size_t)> struct _SFINAE1 {};
    template <class _Up, size_t (_Up::*)() const> struct _SFINAE2 {};
    template <class _Up> static good reserve(_SFINAE1<_Up, &_Up::reserve>*);
    template <class _Up> static bad reserve(...);
    template <class _Up> static good size(_SFINAE2<_Up, &_Up::size>*);
    template <class _Up> static bad size(...);
    static const bool value =
        (sizeof(reserve<_T1>(nullptr)) == sizeof(good) &&
         sizeof(size<_T2>(nullptr)) == sizeof(good));
};

template <class _T1, class _T2>
void try_reserve(_T1&, const _T2&, std::false_type)
{
}

template <class _T1, class _T2>
void try_reserve(_T1& dest, const _T2& src, std::true_type)
{
    dest.reserve(src.size());
}

try_reserve(
    result, inputs,
    std::integral_constant<
        bool, can_reserve<decltype(result), _Rng>::value>());
```

用 Concept 进行简化


```
template <typename _T1, typename _T2>
concept bool can_reserve =
    requires(_T1& dest, const _T2& src) {
        dest.reserve(src.size());
    };

template <class _T1, class _T2>
void try_reserve(_T1&, const _T2&, std::false_type)
{
}

template <class _T1, class _T2>
void try_reserve(_T1& dest, const _T2& src, std::true_type)
{
    dest.reserve(src.size());
}

try_reserve(
    result, inputs,
    std::integral_constant<
        bool, can_reserve<decltype(result), _Rng>>());
```

用 if constexpr 进一步简化


```
template <typename _T1, typename _T2>
concept bool can_reserve =
    requires(_T1& dest, const _T2& src) {
        dest.reserve(src.size());
    };

if constexpr (can_reserve<decltype(result), _Rng>)
{
    result.reserve(inputs.size());
}
```

使用 Concept 的问题

- Concept 的语法可能被进一步改进
  - 如有建议把“concept bool”简化成“concept”
- 特定概念的定义在标准中也可能会变化
  - 目前只有 CMCSTL2 一个提供概念定义的库实现
- 使用概念后能工作的代码可能会停止工作
  - 如果对象使用了特殊技巧，不 Regular 的话
- 概念的出错提示友好性部分被 -Wfatal-errors 抵消

总结

- 更简单
- 更表意
- 更可读
- 只有编译时开销
- 仍未完全成熟，拭目以待 C++20

参考资料

- Bjarne Stroustrup: The C++0x "Remove Concepts" Decision. July 2009.
- Jaakko Järvi, Mat Marcus, and Jacob N. Smith: Programming with C++ concepts. July 2010.
- Tom Honermann: Why Concepts didn't make C++17. March 2016.
- Andrew Sutton: Introducing Concepts. October 2015.
- Andrew Sutton: Defining Concepts. February 2016.
- Eric Niebler and Casey Carter: Working Draft, C++ Extensions for Ranges (Ranges TS). November 2016.
- Richard Smith and James Dennett: Concepts TS revisited. February 2017.
- 我的示例代码：https://github.com/adah1972/cpp_summit_2017

2017 CPP-Summit

备用材料

C++11 时关于 Concept 的争议

- Should a type that meets the requirements of a "concept" automatically be accepted where the "concept" is required (e.g. should a type `X` that provides `+`, `-`, `*`, and `/` with suitable parameters automatically match a "concept" `C` that requires the usual arithmetic operations with suitable parameters) or should an additional explicit statement (a "concept" map from `X` to `C`) that a match is intentional be required?
- Should there be a choice between *automatic* and *explicit* "concepts" and should a designer of a "concept" be able to force every user to follow his choice?
- Should a type `X` that provides a member operation `X::begin()` be considered a match for a "concept" `C<T>` that requires a function `begin(T)` or should a user supply a "concept" map from `T` to `C`? An example is `std::vector` and `std::Range`.

使用 -Wfatal-errors 时的 GCC 出错信息

无 Concept：

```
test_odd.cpp: In instantiation of 'bool odd(N) [with N = double]':
test_odd.cpp:12:25:   required from here
test_odd.cpp:6:19: error: invalid operands of types 'double' and
'int' to binary 'operator&'
     return bool(n & 0x1);
                 ~~^~~~
```

有 Concept：

```
test_odd_concepts.cpp: In function 'int main()':
test_odd_concepts.cpp:14:25: error: cannot call function 'bool
odd(N) requires   Integer<N> [with N = double]'
     std::cout << odd(1.0) << std::endl;  // Does not compile
                         ^
```

概念约束写在函数名后面的情况

使用 Concept 时的 GCC 出错信息


```
test_odd_concepts.cpp: In function 'int main()':
test_odd_concepts.cpp:13:25: error: cannot call function 'bool odd(N) [with N = double]'
     std::cout << odd(1.0) << std::endl;  // Does not compile
                         ^
test_odd_concepts.cpp:5:13: note:   constraints not satisfied
 inline bool odd(N n)
             ^~~
In file included from test_odd_concepts.cpp:2:0:
concepts.h:107:14: note: within 'template<class T> concept const bool Integer<T> [with T = double]'
 concept bool Integer =
              ^~~~~~~
concepts.h:107:14: note:     with 'double a'
concepts.h:107:14: note:     with 'double b'
concepts.h:107:14: note: the required expression '(a % b)' would be ill-formed
concepts.h:107:14: note: the required expression 'a %= b' would be ill-formed
```

概念约束写在其他地方

使用 -Wfatal-errors 时不显示