D.1 chrono头文件

<chrono>头文件作为 `time_point` 的提供者，具有代表时间点的类，duration类和时钟类。每个时钟都有一个 `is_steady` 静态数据成员，这个成员用来表示该时钟是否是一个*稳定的*时钟(以匀速计时的时钟，且不可调节)。 `std::chrono::steady_clock` 是唯一个能保证稳定的时钟类。

头文件正文

```
namespace std
{
  namespace chrono
  {
    template<typename Rep,typename Period = ratio<1>>
    class duration;
    template<
        typename Clock,
        typename Duration = typename Clock::duration>
    class time_point;
    class system_clock;
    class steady_clock;
    typedef unspecified-clock-type high_resolution_clock;
  }
}
```

D.1.1 std::chrono::duration类型模板

`std::chrono::duration` 类模板可以用来表示时间。模板参数 `Rep` 和 `Period` 是用来存储持续时间的数据类型， `std::ratio` 实例代表了时间的长度(几分之一秒)，其表示了在两次“时钟滴答”后的时间(时钟周期)。因此， `std::chrono::duration<int, std::milli>` 即为，时间以毫秒数的形式存储到int类型中，而 `std::chrono::duration<short, std::ratio<1,50>>` 则是记录1/50秒的个数，并将个数存入short类型的变量中，还有 `std::chrono::duration <long long, std::ratio<60,1>>` 则是将分钟数存储到long long类型的变量中。

类的定义

```
template <class Rep, class Period=ratio<1> >
class duration
{
public:
  typedef Rep rep;
  typedef Period period;

  constexpr duration() = default;
  ~duration() = default;

  duration(const duration&) = default;
  duration& operator=(const duration&) = default;

  template <class Rep2>
  constexpr explicit duration(const Rep2& r);

  template <class Rep2, class Period2>
  constexpr duration(const duration<Rep2, Period2>& d);

  constexpr rep count() const;
  constexpr duration operator+() const;
  constexpr duration operator-() const;

  duration& operator++();
  duration operator++(int);
  duration& operator--();
  duration operator--(int);

  duration& operator+=(const duration& d);
  duration& operator-=(const duration& d);
  duration& operator*=(const rep& rhs);
  duration& operator/=(const rep& rhs);

  duration& operator%=(const rep& rhs);
  duration& operator%=(const duration& rhs);

  static constexpr duration zero();
  static constexpr duration min();
  static constexpr duration max();
};

template <class Rep1, class Period1, class Rep2, class Period2>
```

```
constexpr bool operator==(
    const duration<Rep1, Period1>& lhs,
    const duration<Rep2, Period2>& rhs);

template <class Rep1, class Period1, class Rep2, class Period2>
    constexpr bool operator!=(
    const duration<Rep1, Period1>& lhs,
    const duration<Rep2, Period2>& rhs);

template <class Rep1, class Period1, class Rep2, class Period2>
    constexpr bool operator<(
    const duration<Rep1, Period1>& lhs,
    const duration<Rep2, Period2>& rhs);

template <class Rep1, class Period1, class Rep2, class Period2>
    constexpr bool operator<=(
    const duration<Rep1, Period1>& lhs,
    const duration<Rep2, Period2>& rhs);

template <class Rep1, class Period1, class Rep2, class Period2>
    constexpr bool operator>(
    const duration<Rep1, Period1>& lhs,
    const duration<Rep2, Period2>& rhs);

template <class Rep1, class Period1, class Rep2, class Period2>
    constexpr bool operator>=(
    const duration<Rep1, Period1>& lhs,
    const duration<Rep2, Period2>& rhs);

template <class ToDuration, class Rep, class Period>
    constexpr ToDuration duration_cast(const duration<Rep, Period>& d);
```

要求

`Rep` 必须是内置数值类型，或是自定义的类数值类型。

`Period` 必须是 `std::ratio<>` 实例。

std::chrono::duration::Rep 类型

用来记录 `dration` 中时钟周期的数量。

声明

```
typedef Rep rep;
```

`rep` 类型用来记录 `duration` 对象内部的表示。

std::chrono::duration::Period 类型

这个类型必须是一个 `std::ratio` 的特化实例，用来表示在继续时间中，1s所要记录的次数。例如，当 `period` 是 `std::ratio<1, 50>` ， `duration` 变量的count()就会在N秒钟返回50N。

声明

```
typedef Period period;
```

std::chrono::duration 默认构造函数

使用默认值构造 `std::chrono::duration` 实例

声明

```
constexpr duration() = default;
```

效果

`duration` 内部值(例如 `rep` 类型的值)都已初始化。

std::chrono::duration 需要计数值的转换构造函数

通过给定的数值来构造 `std::chrono::duration` 实例。

声明

```
1 template <class Rep2>;
2 constexpr explicit duration(const Rep2& r);
```

效果

`duration` 对象的内部值会使用 `static_cast<rep>(r)` 进行初始化。

结果

当Rep2隐式转换为Rep，Rep是浮点类型或Rep2不是浮点类型，这个构造函数才能使用。

后验条件

```
this->count()==static_cast<rep>(r)
```

std::chrono::duration 需要另一个std::chrono::duration值的转化构造函数

通过另一个 `std::chrono::duration` 类实例中的计数值来构造一个 `std::chrono::duration` 类实例。

声明

```
template <class Rep2, class Period>
constexpr duration(const duration<Rep2,Period2>& d);
```

结果

duration对象的内部值通过 `duration_cast<duration<Rep,Period>>(d).count()` 初始化。

要求

当Rep是一个浮点类或Rep2不是浮点类，且Period2是Period数的倍数(比如，ratio_divide<Period2,Period>::den==1)时，才能调用该重载。当一个较小的数据转换为一个较大的数据时，使用该构造函数就能避免数位截断和精度损失。

后验条件

```
this->count() == dutation_cast&lt;duration<Rep, Period>>(d).count()
```

例子

```
duration<int, ratio<1, 1000>> ms(5);  // 5毫秒
duration<int, ratio<1, 1>> s(ms);  // 错误：不能将ms当做s进行存储
duration<double, ratio<1,1>> s2(ms);  // 合法：s2.count() == 0.005
duration<int, ration<1, 1000000>> us<ms>;  // 合法:us.count() == 5000
```

std::chrono::duration::count 成员函数

查询持续时长。

声明

```
constexpr rep count() const;
```

返回

返回duration的内部值，其值类型和rep一样。

std::chrono::duration::operator+ 加法操作符

这是一个空操作：只会返回*this的副本。

声明

```
constexpr duration operator+() const;
```

返回 `*this`

std::chrono::duration::operator- 减法操作符

返回将内部值只为负数的*this副本。

声明

```
constexpr duration operator-() const;
```

返回 `duration(--this->count());`

std::chrono::duration::operator++ 前置自加操作符

增加内部计数值。

声明

```
duration& operator++();
```

结果

```
++this->internal_count;
```

返回 `*this`

std::chrono::duration::operator++ 后置自加操作符

自加内部计数值，并且返回还没有增加前的*this。

声明

```
duration operator++(int);
```

结果

```
1 duration temp(*this);
2 ++(*this);
3 return temp;
```

std::chrono::duration::operator-- 前置自减操作符

自减内部计数值

声明

```
duration& operator--();
```

结果

```
--this->internal_count;
```

返回 `*this`

std::chrono::duration::operator-- 前置自减操作符

自减内部计数值，并且返回还没有减少前的*this。

声明

```
duration operator--(int);
```

结果

```
duration temp(*this);
--(*this);
return temp;
```

std::chrono::duration::operator+= 复合赋值操作符

将其他duration对象中的内部值增加到现有duration对象当中。

声明

```
duration& operator+=(duration const& other);
```

结果

```
internal_count+=other.count();
```

返回 `*this`

std::chrono::duration::operator-= 复合赋值操作符

现有duration对象减去其他duration对象中的内部值。

声明

```
duration& operator-=(duration const& other);
```

结果

```
internal_count-=other.count();
```

返回 `*this`

std::chrono::duration::operator*= 复合赋值操作符

内部值乘以一个给定的值。

声明

```
duration& operator*=(rep const& rhs);
```

结果

```
internal_count*=rhs;
```

返回 `*this`

std::chrono::duration::operator/= 复合赋值操作符

内部值除以一个给定的值。

声明

```
duration& operator/=(rep const& rhs);
```

结果

```
internal_count/=rhs;
```

返回 `*this`

std::chrono::duration::operator%= 复合赋值操作符

内部值对一个给定的值求余。

声明

```
duration& operator%=(rep const& rhs);
```

结果

```
internal_count%=rhs;
```

返回 `*this`

std::chrono::duration::operator%= 复合赋值操作符(重载)

内部值对另一个duration类的内部值求余。

声明

```
duration& operator%=(duration const& rhs);
```

结果

```
internal_count%=rhs.count();
```

返回 `*this`

std::chrono::duration::zero 静态成员函数

返回一个内部值为0的duration对象。

声明

```
constexpr duration zero();
```

返回

```
duration(duration_values<rep>::zero());
```

std::chrono::duration::min 静态成员函数

返回duration类实例化后能表示的最小值。

声明

```
constexpr duration min();
```

返回

```
duration(duration_values<rep>::min());
```

std::chrono::duration::max 静态成员函数

返回duration类实例化后能表示的最大值。

声明

```
constexpr duration max();
```

返回

```
duration(duration_values<rep>::max());
```

std::chrono::duration 等于比较操作符

比较两个duration对象是否相等。

声明

```
template <class Rep1, class Period1, class Rep2, class Period2>
constexpr bool operator==(
const duration<Rep1, Period1>& lhs,
const duration<Rep2, Period2>& rhs);
```

要求

`lhs` 和 `rhs` 两种类型可以互相进行隐式转换。当两种类型无法进行隐式转换，或是可以互相转换的两个不同类型的duration类，则表达式不合理。

结果

当 `CommonDuration` 和 `std::common_type< duration< Rep1, Period1>, duration< Rep2, Period2>>::type` 同类，那么 `lhs==rhs` 就会返回 `CommonDuration(lhs).count()==CommonDuration(rhs).count()` 。

std::chrono::duration 不等于比较操作符

比较两个duration对象是否不相等。

声明

```
template <class Rep1, class Period1, class Rep2, class Period2>
constexpr bool operator!=(
   const duration<Rep1, Period1>& lhs,
   const duration<Rep2, Period2>& rhs);
```

要求

`lhs` 和 `rhs` 两种类型可以互相进行隐式转换。当两种类型无法进行隐式转换，或是可以互相转换的两个不同类型的duration类，则表达式不合理。

返回 `!(lhs==rhs)`

std::chrono::duration 小于比较操作符

比较两个duration对象是否小于。

声明

```
template <class Rep1, class Period1, class Rep2, class Period2>
constexpr bool operator<(
   const duration<Rep1, Period1>& lhs,
   const duration<Rep2, Period2>& rhs);
```

要求

`lhs` 和 `rhs` 两种类型可以互相进行隐式转换。当两种类型无法进行隐式转换，或是可以互相转换的两个不同类型的duration类，则表达式不合理。

结果

当 CommonDuration 和 std::common_type< duration< Rep1, Period1>, duration< Rep2, Period2>>::type 同类，那么 lhs<rhs 就会返回 CommonDuration(lhs).count()<CommonDuration(rhs).count()。

std::chrono::duration 大于比较操作符

比较两个duration对象是否大于。

声明

```
template <class Rep1, class Period1, class Rep2, class Period2>
constexpr bool operator>(
   const duration<Rep1, Period1>& lhs,
   const duration<Rep2, Period2>& rhs);
```

要求

lhs 和 rhs 两种类型可以互相进行隐式转换。当两种类型无法进行隐式转换，或是可以互相转换的两个不同类型的duration类，则表达式不合理。

返回 rhs<lhs

std::chrono::duration 小于等于比较操作符

比较两个duration对象是否小于等于。

声明

```
template <class Rep1, class Period1, class Rep2, class Period2>
constexpr bool operator<=(
   const duration<Rep1, Period1>& lhs,
   const duration<Rep2, Period2>& rhs);
```

要求

lhs 和 rhs 两种类型可以互相进行隐式转换。当两种类型无法进行隐式转换，或是可以互相转换的两个不同类型的duration类，则表达式不合理。

返回 `!(rhs<lhs)`

std::chrono::duration 大于等于比较操作符

比较两个duration对象是否大于等于。

声明

```
template <class Rep1, class Period1, class Rep2, class Period2>
constexpr bool operator>=(
   const duration<Rep1, Period1>& lhs,
   const duration<Rep2, Period2>& rhs);
```

要求

`lhs` 和 `rhs` 两种类型可以互相进行隐式转换。当两种类型无法进行隐式转换，或是可以互相转换的两个不同类型的duration类，则表达式不合理。

返回 `!(lhs<rhs)`

std::chrono::duration_cast 非成员函数

显示将一个 `std::chrono::duration` 对象转化为另一个 `std::chrono::duration` 实例。

声明

```
template <class ToDuration, class Rep, class Period>
constexpr ToDuration duration_cast(const duration<Rep, Period>& d);
```

要求

ToDuration必须是 `std::chrono::duration` 的实例。

返回

duration类d转换为指定类型ToDuration。这种方式可以在不同尺寸和表示类型的转换中尽可能减少精度损失。

D.1.2 std::chrono::time_point类型模板

`std::chrono::time_point` 类型模板通过(特别的)时钟来表示某个时间点。这个时钟代表的是从epoch(1970-01-01 00:00:00 UTC，作为UNIX系列系统的特定时间戳)到现在的时间。模板参数Clock代表使用的使用(不同的使用必定有自己独特的类型)，而Duration模板参数使用来测量从epoch到现在的时间，并且这个参数的类型必须是 `std::chrono::duration` 类型。Duration默认存储Clock上的测量值。

类型定义

```
template <class Clock,class Duration = typename Clock::duration>
class time_point
{
public:
  typedef Clock clock;
  typedef Duration duration;
  typedef typename duration::rep rep;
  typedef typename duration::period period;

  time_point();
  explicit time_point(const duration& d);

  template <class Duration2>
  time_point(const time_point<clock, Duration2>& t);

  duration time_since_epoch() const;

  time_point& operator+=(const duration& d);
  time_point& operator-=(const duration& d);

  static constexpr time_point min();
  static constexpr time_point max();
};
```

std::chrono::time_point 默认构造函数

构造time_point代表着，使用相关的Clock，记录从epoch到现在的时间；其内部计时使用Duration::zero()进行初始化。

声明

```
  time_point();
```

后验条件

对于使用默认构造函数构造出的time_point对象tp，

`tp.time_since_epoch() == tp::duration::zero()` 。

std::chrono::time_point 需要时间长度的构造函数

构造time_point代表着，使用相关的Clock，记录从epoch到现在的时间。

声明

```
explicit time_point(const duration& d);
```

后验条件

当有一个time_point对象tp，是通过duration d构造出来的(tp(d))，那么

`tp.time_since_epoch() == d` 。

std::chrono::time_point 转换构造函数

构造time_point代表着，使用相关的Clock，记录从epoch到现在的时间。

声明

```
template <class Duration2>
time_point(const time_point<clock, Duration2>& t);
```

要求

Duration2必须呢个隐式转换为Duration。

效果

当 `time_point(t.time_since_epoch())` 存在，从t.time_since_epoch()中获取的返回值，可以隐式转换成Duration类型的对象，并且这个值可以存储在一个新的time_point对象中。

(扩展阅读：as-if准则)

std::chrono::time_point::time_since_epoch 成员函数

返回当前time_point从epoch到现在的具体时长。

声明

```
duration time_since_epoch() const;
```

返回
duration的值存储在*this中。

std::chrono::time_point::operator+= 复合赋值函数

将指定的duration的值与原存储在指定的time_point对象中的duration相加，并将加后值存储在*this对象中。

声明

```
time_point& operator+=(const duration& d);
```

效果
将d的值和duration对象的值相加，存储在*this中，就如同this->internal_duration += d;

返回 `*this`

std::chrono::time_point::operator-= 复合赋值函数

将指定的duration的值与原存储在指定的time_point对象中的duration相减，并将加后值存储在*this对象中。

声明

```
time_point& operator-=(const duration& d);
```

效果
将d的值和duration对象的值相减，存储在*this中，就如同this->internal_duration -= d;

返回 `*this`

std::chrono::time_point::min 静态成员函数

获取time_point对象可能表示的最小值。

声明

```
static constexpr time_point min();
```

返回

```
time_point(time_point::duration::min()) (see 11.1.1.15)
```

std::chrono::time_point::max 静态成员函数

获取time_point对象可能表示的最大值。

声明

```
static constexpr time_point max();
```

返回

```
time_point(time_point::duration::max()) (see 11.1.1.16)
```

D.1.3 std::chrono::system_clock类

`std::chrono::system_clock` 类提供给了从系统实时时钟上获取当前时间功能。可以调用 `std::chrono::system_clock::now()` 来获取当前的时间。`std::chrono::system_clock::time_point` 也可以通过 `std::chrono::system_clock::to_time_t()` 和 `std::chrono::system_clock::to_time_point()` 函数返回值转换成time_t类型。系统时钟不稳定，所以 `std::chrono::system_clock::now()` 获取到的时间可能会早于之前的一次调用(比如，时钟被手动调整过或与外部时钟进行了同步)。

类型定义

```
class system_clock
{
public:
  typedef unspecified-integral-type rep;
  typedef std::ratio<unspecified,unspecified> period;
  typedef std::chrono::duration<rep,period> duration;
  typedef std::chrono::time_point<system_clock> time_point;
  static const bool is_steady=unspecified;

  static time_point now() noexcept;

  static time_t to_time_t(const time_point& t) noexcept;
  static time_point from_time_t(time_t t) noexcept;
};
```

std::chrono::system_clock::rep 类型定义

将时间周期数记录在一个duration值中

声明

```
typedef unspecified-integral-type rep;
```

std::chrono::system_clock::period 类型定义

类型为 `std::ratio` 类型模板，通过在两个不同的duration或time_point间特化最小秒数(或将1秒分为好几份)。period指定了时钟的精度，而非时钟频率。

声明

```
typedef std::ratio<unspecified,unspecified> period;
```

std::chrono::system_clock::duration 类型定义

类型为 `std::ratio` 类型模板，通过系统实时时钟获取两个时间点之间的时长。

声明

```
typedef std::chrono::duration<
   std::chrono::system_clock::rep,
   std::chrono::system_clock::period> duration;
```

std::chrono::system_clock::time_point 类型定义

类型为 std::ratio 类型模板，通过系统实时时钟获取当前时间点的时间。

声明

```
typedef std::chrono::time_point&lt;std::chrono::system_clock&gt; time_point;
```

std::chrono::system_clock::now 静态成员函数

从系统实时时钟上获取当前的外部设备显示的时间。

声明

```
time_point now() noexcept;
```

返回
time_point类型变量来代表当前系统实时时钟的时间。

抛出
当错误发生， std::system_error 异常将会抛出。

std::chrono::system_clock:to_time_t 静态成员函数

将time_point类型值转化为time_t。

声明

```
time_t to_time_t(time_point const& t) noexcept;
```

返回
通过对t进行舍入或截断精度，将其转化为一个time_t类型的值。

抛出

当错误发生， `std::system_error` 异常将会抛出。

std::chrono::system_clock::from_time_t 静态成员函数

声明

```
time_point from_time_t(time_t const& t) noexcept;
```

返回

time_point中的值与t中的值一样。

抛出

当错误发生， `std::system_error` 异常将会抛出。

D.1.4 std::chrono::steady_clock类

`std::chrono::steady_clock` 能访问系统稳定时钟。可以通过调用 `std::chrono::steady_clock::now()` 获取当前的时间。设备上显示的时间，与使用 `std::chrono::steady_clock::now()` 获取的时间没有固定的关系。稳定时钟是无法回调的，所以在 `std::chrono::steady_clock::now()` 两次调用后，第二次调用获取的时间必定等于或大于第一次获得的时间。时钟以固定的速率进行计时。

类型定义

```
1  class steady_clock
2  {
3  public:
4    typedef unspecified-integral-type rep;
5    typedef std::ratio<
6        unspecified,unspecified> period;
7    typedef std::chrono::duration<rep,period> duration;
8    typedef std::chrono::time_point<steady_clock>
9        time_point;
10   static const bool is_steady=true;
11
12   static time_point now() noexcept;
```

```
};
```

std::chrono::steady_clock::rep 类型定义

定义一个整型，用来保存duration的值。

声明

```
typedef unspecified-integral-type rep;
```

std::chrono::steady_clock::period 类型定义

类型为 `std::ratio` 类型模板，通过在两个不同的duration或time_point间特化最小秒数(或将1秒分为好几份)。period指定了时钟的精度，而非时钟频率。

声明

```
typedef std::ratio<unspecified,unspecified> period;
```

std::chrono::steady_clock::duration 类型定义

类型为 `std::ratio` 类型模板，通过系统实时时钟获取两个时间点之间的时长。

声明

```
typedef std::chrono::duration<
   std::chrono::system_clock::rep,
   std::chrono::system_clock::period> duration;
```

std::chrono::steady_clock::time_point 类型定义

`std::chrono::time_point` 类型实例，可以存储从系统稳定时钟返回的时间点。

声明

```
typedef std::chrono::time_point<std::chrono::steady_clock> time_point;
```

std::chrono::steady_clock::now 静态成员函数

从系统稳定时钟获取当前时间。

声明

```
time_point now() noexcept;
```

返回
time_point表示当前系统稳定时钟的时间。

抛出
当遇到错误，会抛出 std::system_error 异常。

同步
当先行调用过一次 std::chrono::steady_clock::now() ，那么下一次time_point获取的值，一定大于等于第一次获取的值。

D.1.5 std::chrono::high_resolution_clock类定义

td::chrono::high_resolution_clock 类能访问系统高精度时钟。和所有其他时钟一样，通过调用 std::chrono::high_resolution_clock::now() 来获取当前时间。
std::chrono::high_resolution_clock 可能是 std::chrono::system_clock 类或 std::chrono::steady_clock 类的别名，也可能就是独立的一个类。

通过 std::chrono::high_resolution_clock 具有所有标准库支持时钟中最高的精度，这就意味着使用 std::chrono::high_resolution_clock::now() 要花掉一些时间。所以，当你再调用 std::chrono::high_resolution_clock::now() 的时候，需要注意函数本身的时间开销。

类型定义

```
class high_resolution_clock
{
public:
  typedef unspecified-integral-type rep;
  typedef std::ratio<
      unspecified,unspecified> period;
```

```
  typedef std::chrono::duration<rep,period> duration;
  typedef std::chrono::time_point<
      unspecified> time_point;
  static const bool is_steady=unspecified;

  static time_point now() noexcept;
};
```