tuple

用处：简化定义结构体，懒得定义结构体的场景。

tuple是更加泛化的pair。特点如下

- 是模板
- 成员任意
- 各个成员类型可以不同
  也就是，tuple就是一组数据，可以看成一个结构体。定义在tuple头文件。

定义

定义方法如下：

```
tuple<size_t,size_t,size_t> threeD; // 三个成员都被值初始化为0
// 相当于结构体：
struct threeD{
    size_t a;
    size_t b;
    size_t c;
};
tuple<string,vector<dobule>,int,list<int>>
    someVal("constants",{3.14,3.14},42,{0,1,2});
```

第一个使用了tuple的默认构造函数，对每个成员进行值初始化。第二个使用参数为每个成员提供一个初始值来构造，且该构造函数时explicit的，也就是我们必须使用直接初始化语法：

```
tuple<size_t,size_t,size_t> threeD={1,2,3}; // 错误，{1,2,3}不可隐式转化为tuple
tuple<size_t,size_t,size_t> threeD{1,2,3};  // 直接初始化，正确
```

类似make_pair，可以使用make_tupe生成tuple对象：

```
auto iter=make_tuple("0-999-7835-X",2,2.000);
```

make_tuple根据初始值的类型推断tuple的类型。这里item是一个类型为tuple<const char *,int,double>的tuple。

tuple成员的访问

pair的访问，使用first和second成员。访问tuple使用get

```
auto book=get<0> (item);     // 返回item第一个成员
auto cnt=get<1> (item);      // 返回item第二个成员
get<2>(item)*=0.8;           // 可以修改tupled的成员
```

此外，一些辅助信息，

```
typedef decltype(item) trans;   // trans时item类型
// using trans=decltyp(item);
// 返回trans类型对象中的成员数量
size_t sz=tuple_size<trans>::value; // 返回3
// cnt的类型和item中的第二个成员相同
tuple_element<1,trans>::type cnt=get<1>(item);  // cnt是一个int
```

tuple_size和tuple_element都是模板，其中value是tuple_size的public static数据成员。

关系和相等运算符

关系和相等运算符逐对比较左侧的tuple和右侧的tuple的成员。只有两个tuple具有相同数量的成员时，我们才可以比较。当然，每个成员使用==运算符或关系运算符应该是合法的。

```
tuple<string,string> t1("1","2");
tuple<size_t,size_t> t2(1,2);
bool b=(t1==t2);     // 错误，不可以比较size_t和string
```

> Notes: tuple定义了<和==运算符，所以我们可以将tuple序列传递给有关算法，并且在无序容器中将tuple作为关键字类型。

---

tuple作用之一，返回多个值

```
using _Type=tuple<_Ty1,_Ty2>;
vector<_Type> foo(){
    vector<_Type> ret;
    ret.push_back(make_tuple(/*...*/));
    return ret;
}
```