check 5

3. Implementing the Router

实现一个Router类

- 实现追踪路由表（转发规则或路由列表）
- 转发接收的数据报（到下一跳ip在正确的网络接口上）

```
void add_route(uint32_t route_prefix,                    //ip
               uint8_t prefix_length,                    //掩码
               optional<Address> next_hop,               //下一跳
               size_t interface_num);
```

添加路由

```
void route();
```

从适当网络接口发出数据报到下一跳；实现路由的最长前缀匹配逻辑

- 目标地址最长前缀有效位数和路由最长前缀有效位相同。
- 匹配路由中，选择长度最长的
- 无匹配路由，抛弃数据
- 路由每一条会减少TTL，TTL清零则抛弃数据

router.hh

```
struct route_data{
  std::optional<Address> next_hop{};
  size_t interface_num{};
};

struct hash_pair
{
  template<class T1,class T2>
  size_t operator()(const std::pair<T1,T2> &p) const
  {
    auto hash1=std::hash<T1>{}(p.first);
```

```
    auto hash2=std::hash<T2>{}(p.second);
    return hash1^hash2;
  }
};
class Router
{
  // The router's collection of network interfaces
  std::vector<AsyncNetworkInterface> interfaces_ {};
  //pair中存放route_prefix 和prefix_length
  std::unordered_map<std::pair<uint32_t,uint8_t>,route_data,hash_pair>
route_table{};
public:
  ...
```

router.cc

```
void Router::add_route( const uint32_t route_prefix,
                        const uint8_t prefix_length,
                        const optional<Address> next_hop,
                        const size_t interface_num )
{
  cerr << "DEBUG: adding route " << Address::from_ipv4_numeric( route_prefix ).
       << static_cast<int>( prefix_length ) << " => " << ( next_hop.has_value()
>ip() : "(direct)" )
       << " on interface " << interface_num << "\n";

 route_table.emplace(make_pair(route_prefix,prefix_length),route_data{next_hop,

  (void)route_prefix;
  (void)prefix_length;
  (void)next_hop;
  (void)interface_num;
}

void Router::route()
{
  //遍历网络接口
  for(auto&& interface:interfaces_){
    std::optional<InternetDatagram> ip_dgram;
    while(true)
    {
      //取到ip数据
      ip_dgram=interface.maybe_receive();
      if(ip_dgram==nullopt)
        break;
      //ttl大于0，要减少
      if(ip_dgram.value().header.ttl>0)
        ip_dgram.value().header.ttl--;
```

```
      //ttl为0，丢弃
      if(ip_dgram.value().header.ttl<=0)
        continue;
      ip_dgram.value().header.compute_checksum();

      //start match
      bool is_match_route=false;
      std::pair<std::pair<uint32_t,uint8_t>,route_data> route_best;
      if(route_table.size()==0) continue;
      //是否有默认路由
      bool has_route_defult=false;
      std::pair<std::pair<uint32_t,uint8_t>,route_data> route_defult;

      //遍历路由表
      for(auto&& route:route_table){
        //后缀长
        uint8_t len=32-route.first.second;
        //全0，为默认路由
        if(len==32){
          has_route_defult=true;
          route_defult=route;
          continue;
        }
        //路由前缀相同
        if((route.first.first>>len)==(ip_dgram->header.dst>>len)){
          //最长前缀
          if(route.first.second>=route_best.first.second){
            is_match_route=true;
            route_best=route;
          }
        }
      }
      //未匹配，转发至默认路由
      if(!is_match_route){
        if(has_route_defult){
          route_best=route_defult;
          is_match_route=true;
        }else{
          continue;
        }
      }
      //转发数据
      interfaces_[route_best.second.interface_num]
        .send_datagram(ip_dgram.value(),
route_best.second.next_hop.value_or(Address::from_ipv4_numeric(ip_dgram.value()
    }
  }
}
```

```
Test project /home/sgt/cs/minnow/build
    Start  1: compile with bug-checkers
1/3 Test  #1: compile with bug-checkers ........   Passed    4.90 sec
    Start 35: net_interface
2/3 Test #35: net_interface ....................   Passed    0.03 sec
    Start 36: router
3/3 Test #36: router ...........................   Passed    0.06 sec

100% tests passed, 0 tests failed out of 3

Total Test time (real) =   4.99 sec
Built target check5
```