Morden cpp notes — serialization

Wednesday, December 9, 2020 2:03 PM

JSON

Json起源于JavaScript，一般用Json交换数据，不会太在意性能，推荐Json for Morden C++库，仅需一个头文件json.hpp.

Eg:

```
using json_t = nlohmann::json; //给这个类起个别名:
json_t j;  // JSON对象.
j["age"] = 23;
j["name"] = "spiderman";
j["gear"]["suits"] = "2099";  //"gear":{"suits":"2099"}
j["jobs"] = {"superhero"};
vector<int> v = {1, 2, 3};
j["numbers"] = v;
map<string, int> m = {{"one", 1}, {"two", 2}};
j["kv"] = m;
```

添加完后，用成员函数dump()就可以序列化，得到Json文本形式，默认格式是紧凑输出，没有缩进，如果想易阅读，可以加上指示缩进的参数:

```
cout << j.dump() << endl; //序列化，无缩进.
cout << j.dump(2) << endl; //序列化，缩进2个空格.
```

json_t的反序列化只需调用静态成员函数parse()，得到JSON对象，而且可以用auto自动推导类型

```
string str = R"({  // JSON文本，原始字符串.
    "name": "peter",
    "age": 23,
    "married": true
})";
auto j = json_t::parse(str); // 从字符串反序列化.
assert(j["age"] == 23); //验证结果
assert(j["name"] == "peter");
```

json_t使用异常来处理解析时可能发生的错误，如果不能保证JSON数据的完整，就用try-catch保护，防止错误数据导致程序崩溃.

```
auto txt = "bad:data"s; //不是正确的JSON数据.
try
{
  auto j = json_t::parse(txt); // 从字符串反序列化.
}
catch (std::exception& e)  //捕获异常
{
  cout << e.what() << endl;
}
```

MessagePack

不是纯文本，而是二进制，所以比JSON更小巧，不过没有JSON那么直观.
著名的应用: Redis, Pinterest.
msgpack-c也是仅包含头文件的库，header only，只要包含msgpack.hpp就行了，

```
vector<int> v = {1,2,3,4,5};
msgpack::sbuffer sbuf;    //输出缓冲区
msgpack::pack(sbuf, v);  //序列化.
```

sbuffer是个简单的缓冲区，是对字符串数组的封装，和vector<char>很像，也可以用data()和size()

它的用法不像JSON那么直观，必须同时传递序列化的输出目标和被序列化的对象.

```
cout << sbuf.size() << endl; //查看序列化后数据的长度.
```

函数unpack()反序列化数据，得到的是object_handle，再调用get()，就是object:

```
auto handle = msgpack::unpack(  //反序列化.
                  sbuf.data(), sbuf.size()); // 输入二进制数据.
auto obj = handle.get();   //得到反序列化对象.
vector<int> v2; //必须知道原始数据的类型才能转换还原.
obj.convert(v2);
assert(std::equal(
       begin(v), end(v), begin(v2)));
```

MessagePack又提供了packer类，实现串联的序列化操作，

```
msgpack::sbuffer sbuf; //输出缓冲区.
msgpack::packer<decltype(sbuf)> packer(sbuf); //专门的序列化对象.
packer.pack(10).pack("cyberpunk"s)   //连续序列化多个数据.
      .pack(vector<int>{1,2,3});
```

反序列化:

```
for(decltype(sbuf.size()) offset = 0; //初始偏移量是0
    offset != sbuf.size();){               //直至反序列化结束.

    auto handle = msgpack::unpack(     //反序列化
        sbuf.data(), sbuf.size(), offset);   // 输入二进制数据和偏移量
   auto obj = handle.get();   //得到反序列化的对象.
}
```

使用MSGPACK_DEFINE宏:

```
class Book final
{
 public:
    int     id;
    string  title;
    set<string>  tags;
public:
    MSGPACK_DEFINE(id, title, tags); //实现序列化功能的宏.
};
```

```
Book book1 = {1, "Ready Player One", {"a", "b"}};
msgpack::sbuffer sbuf;  //输出缓冲区.
msgpack::pack(sbuf, book1);  //序列化.
auto obj = msgpack::unpack( //反序列化.
               sbuf.data(), sbuf.size()).get(); //得到反序列化对象.
Book book2;
obj.convert(book2);  //转换反序列化的数据.
assert(book2.id == book1.id);
assert(book2.tags.size() == 2);
cout << book2.title << endl;
```

使用MessagePack时，也要注意数据不完整的情况，用try-catch保护代码.

```
auto txt = ""s;
try
{
  auto handle = msgpack::unpack( // 反序列化
        txt.data(), txt.size());
}
catch (std::exception& e) //捕获异常
{
  cout << e.what() << endl;
}
```

ProtoBuffer

Google出品，是二进制的数据格式，需要安装预处理器和开发库，编译时还要链接动态库-lprotobuf.
特点是数据有模式schema，必须先写一个IDL (Interface description Language) 文件，定义好数据结构.

```
syntax = "proto2"; //使用第2版.
package sample;  //定义命名空间.
message Vendor   //定义消息.
{
  required  uint32  id = 1; // required表示必须字段.
  required  string  name = 2; //有int32/string等基本类型.
  required  bool  valid = 3; // 需要指定字段的序号，序列化时用.
  optional  string  tel = 4; // optional字段可以没有.
}
```

有了接口定义文件，需要再用protoc工具生成对应的C++代码，然后把生成文件加入自己的项目中:

```
protoc --cpp_out=. sample.proto
```

Eg.

```
using vendor_t = sample::Vendor; //类型别名.
vendor_t v; //声明一个对象.
assert(!v.IsInitialized()); // required等字段未初始化.
v.set_id(1); //设置每个字段的值.
v.set_name("sean");
v.set_valid(true);
assert(v.IsInitialized()); // required字段都设置了，数据完整.
assert(v.has_id() && v.id() == 1);
assert(v.has_name() && v.name() == "sean");
assert(v.has_valid() && v.valid());
cout << v.DebugString() << endl; //输出调试字符串
string enc;
v.SerializeToString(&enc); //序列化到字符串
vendor_t v2;
assert(!v2.IsInitialized());
```

Tips:

- JSON是纯文本，容易阅读，方便编辑，适用性最广，
- MessagePack是二进制，小巧高效，
- ProtoBuffer注重安全和性能，大公司常用.