Table of Contents

序

日知录是为效法顾炎武的《日知录》而做，里面会记录自己在学习与工作中学到的东西，该书的目的是为了打造一本属于自己的百科全书，也是自己思想体系的体现。最终里面的每一章可能代表一个方向与学科，比如C++，C#最终会成为一章，经济，中国史会成为一章。如果经济学里面记录的内容不多，就编成一章，如果内容多了，则会编成多章，比如宏观经济学，微观经济学，计量经济学。这样，最终一门大的学科，比如，经济学，就编成了一卷。

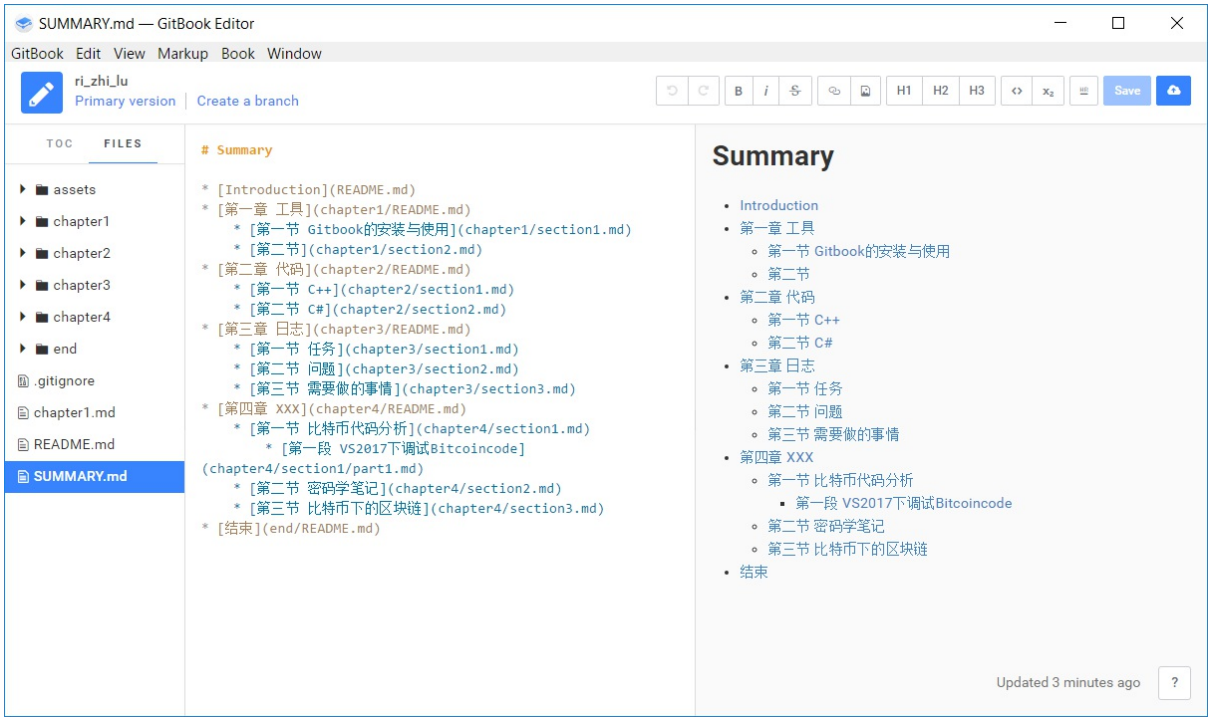
第一章:工具

第一节 Gitbook的安装与使用

Gitbook是Github旗下的产品，它提供书籍的编写与管理的功能，一个核心特征就是，它管理书也像管理代码一样，可以对数据进行fork,建立新的branch,使得书也可以进行版本迭代。也可以与多人合作，来编辑，更新书籍，适合单人创作或者多人共同创作。怎么利用gitbook生成自己的书，可以参考如下链接 https://www.jianshu.com/p/cf4989c20bd8 https://www.cnblogs.com/Lam7/p/6109872.html

在安装好Node.js与gitbook之后，我们可以利用gitbook的GUI创建一本书，写入一些章节，然后保存，然后在cmd到该书summary.md所在的路径，然后再利用gitbook init进行初始化，然后再利用gitbook serve在浏览器上对书内容进行访问。当然，个gitbook serve这一步会出错，会出现fontsettings.js找不到的情况，我们需要修改C:\Users\pili.gitbook\versions\3.2.3\lib\output\website\copyPluginAssets.js，设置confirm: false，再重启个gitbook serve就可以了。详见https://github.com/GitbookIO/gitbook/issues/1309 然后就可以在 http://localhost:4000 访问书籍了

如果想让章节有层次感的显示，则可以在summary.md中设置



Error: Error with command "svgexport"

在cmd中安装：npm install svgexport -g

第二节 编辑数学公式

mathjax

可以参考https://598753468.gitbooks.io/tex/content/fei_xian_xing_fu_hao.html
数学公式的编辑类似于Latex.

需要安装mathjax

```
npm install mathjax
安装特定版本的npm install mathjax@2.6.1
```

首先在书籍project的最顶成新建一个book.json,内容如下

```
{
    "gitbook": "3.2.3",
    "plugins": ["mathjax"],
    "links": {
        "sidebar": {
            "Contact us / Support": "https://www.gitbook.com/contact"
        }
    },
    "pluginsConfig": {
    "mathjax":{
            "forceSVG": true
        }
        }
}
```

然后再用gitbook install命令安装mathjax

安装之后gitbook就出现编译错误了，也不能编译生成pdf文件。不论mathjax是哪个版本，从2.5开始都出错。下面选择用katex

gitbook pdf ./ mybook.pdf（./ mybook.pdf 之间有空格）

http://ldehai.com/blog/2016/11/30/write-with-gitbook/

no such file fontsettings.js,在上一节有讲怎么处理。

安装Katex

https://github.com/GitbookIO/plugin-katex

1. 在你的书籍文件夹里创建book.json
2. 里面写入如下内容

    ```
    {
        "plugins": ["katex"]
    }
    ```

4. 运行安装 gitbook install就安装好了(安装很慢，一个小时),可以换个地方安装，只要把book.json换个地方，再在这个folder安装gitbook install，然后把node_modules 拷到你书籍所在的folder就可以。

中文在cmd中乱码的问题：

1. 打开cmd, 输入chcp 65001chcp 65001
2. 右击cmd上方，选择属性-->字体-->SimSun-ExtB就可以显示了。

语法高亮

```
Supported languages
This is the list of all 120 languages currently supported by Prism, with their corresponding alias, to use in place of xxxx in the language-xxxx cl
```

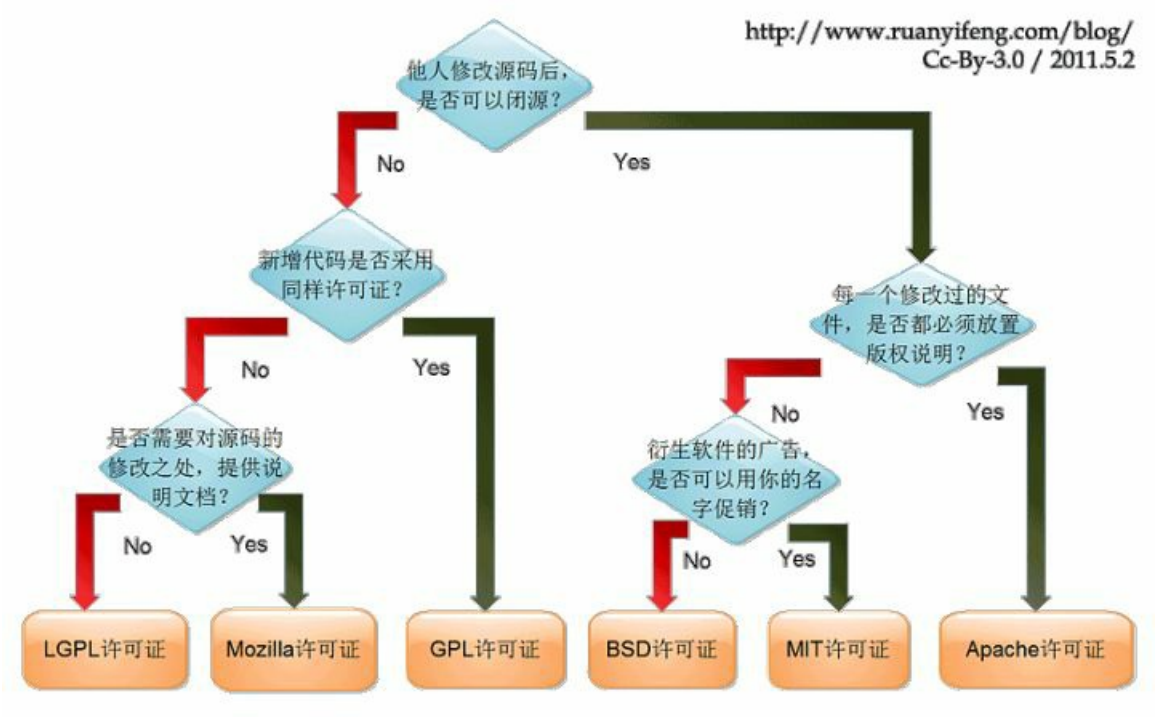
```
ass:



Markup - markup
CSS - css
C-like - clike
JavaScript - javascript
ABAP - abap
ActionScript - actionscript
Ada - ada
Apache Configuration - apacheconf
APL - apl
AppleScript - applescript
AsciiDoc - asciidoc
ASP.NET (C#) - aspnet
AutoIt - autoit
AutoHotkey - autohotkey
Bash - bash
BASIC - basic
Batch - batch
Bison - bison
Brainfuck - brainfuck
Bro - bro
C - c
C# - csharp
C++ - cpp
CoffeeScript - coffeescript
Crystal - crystal
CSS Extras - css-extras
D - d
Dart - dart
Diff - diff
Docker - docker
Eiffel - eiffel
Elixir - elixir
Erlang - erlang
F# - fsharp
Fortran - fortran
Gherkin - gherkin
Git - git
GLSL - glsl
Go - go
GraphQL - graphql
Groovy - groovy
Haml - haml
Handlebars - handlebars
Haskell - haskell
Haxe - haxe
HTTP - http
Icon - icon
Inform 7 - inform7
Ini - ini
J - j
Jade - jade
Java - java
Jolie - jolie
JSON - json
Julia - julia
Keyman - keyman
Kotlin - kotlin
LaTeX - latex
Less - less
LiveScript - livescript
LOLCODE - lolcode
Lua - lua
Makefile - makefile
Markdown - markdown
MATLAB - matlab
MEL - mel
Mizar - mizar
Monkey - monkey
NASM - nasm
nginx - nginx
Nim - nim
Nix - nix
NSIS - nsis
Objective-C - objectivec
OCaml - ocaml
Oz - oz
PARI/GP - parigp
Parser - parser
Pascal - pascal
```

```
Perl - perl
PHP - php
PHP Extras - php-extras
PowerShell - powershell
Processing - processing
Prolog - prolog
.properties - properties
Protocol Buffers - protobuf
Puppet - puppet
Pure - pure
Python - python
Q - q
Qore - qore
R - r
React JSX - jsx
Reason - reason
reST (reStructuredText) - rest
Rip - rip
Roboconf - roboconf
Ruby - ruby
Rust - rust
SAS - sas
Sass (Sass) - sass
Sass (Scss) - scss
Scala - scala
Scheme - scheme
Smalltalk - smalltalk
Smarty - smarty
SQL - sql
Stylus - stylus
Swift - swift
Tcl - tcl
Textile - textile
Twig - twig
TypeScript - typescript
Verilog - verilog
VHDL - vhdl
vim - vim
Wiki markup - wiki
Xojo (REALbasic) - xojo
YAML - yaml
```

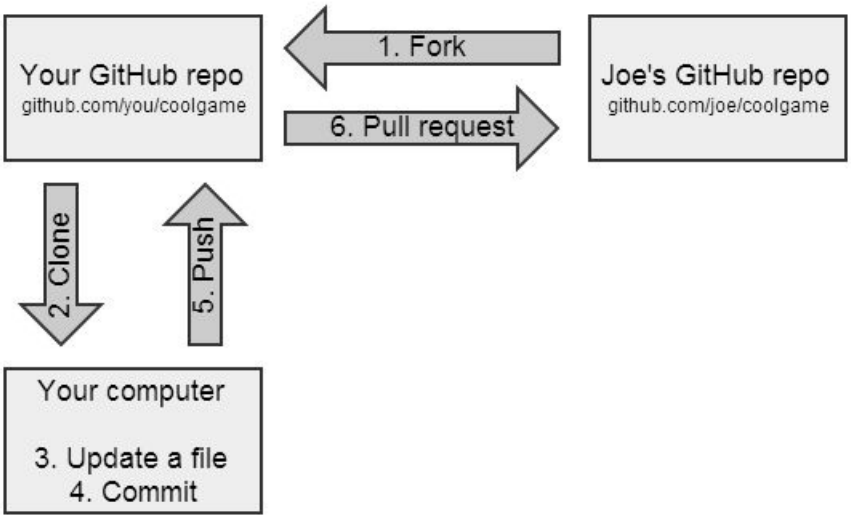
库函数的使用与说明

开源库的使用说明:

如何选择开源许可证？



Github Fork:



Google Test

Google C++ 单元测试框架 核心是添加include与lib路径，以及把运行时库设置成MTd 对工程名右键->属性->配置属性->C/C++->代码生成->运行时库：与前面gtest配置一样，选择MTd；代码位于：C:\Data\ShareFolder\Works\Miura\googletest-master

第二章 设计模式

第一节 工厂模式

第三章 日志

第一节 任务

1. 20180508
    i. Debug MR 以对MR整体架构有个了解，知道哪些是硬件，哪些是与硬件无关
    ii. 中段目的是抽离出一个与Miura机器无关的通用平台
    iii. 最终目的是把RS塞进这个通用平台，也就是把RS建立在这个通用的平台之上
    iv. Boost 库中有uBLAS——Boost 线性代数基础程序库，因此，C++涉及矩阵运算的可以调用这个库。
        i. 自己加工下，但是要考虑跨平台的问题，因此不能封装成dll
2. 20180509
    i. 通过MR提炼出通用的模板（类似于基类），那么通用的模板应该包含哪些东西？这是核心的问题。

第二节 问题项目:

基础线性代数库

1. 2D插值
    i. 基于拟合公式，而拟合公式就是1D插值
    ii. 第一个，实现克里金插值
2. 版本的迭代问题，每次也不求代码很完善，一次次测试与迭代，在过程中追求完美。
    i. 每个版本要说明改进的地方，为啥要改
    ii. https://github.com/lbbc1117/Nacho
    iii. 矩阵求逆需要记下一些notes或者文档。
        i. 这些只是一些练习，真实的可能会调用一些现成的库。目的是练练C++，也是为以后工作备用
    iv. SVD 求解论文
        i. http://people.duke.edu/~hpgavin/SystemID/References/Golub+Reinsch-NM-1970.pdf
    v. **Armadillo** C++ library for linear algebra scientific computing http://arma.sourceforge.net/
        i. https://github.com/conradsnicta/armadillo-code/
        ii. https://en.wikipedia.org/wiki/Comparison_of_linear_algebra_libraries
    vi. 适合自己的才是最好的，
        i. 做一个自己实用的线性代数库，尽可能能供多的人去继承，去开发。
        ii. 要有自己的开发文档。
    vii. eigen：https://github.com/Li-Simon/eigen-git-mirror
        i. 功能比**Armadillo** 强大

第三节 需要做的事情

201805

20180508

- [ ] 怎么把自己用gitbook创作的书同步到网上，这样就可以方便查看了，最好能设置只能个人看。
- [ ] C# Lamda表达式还是需要多看一些，自己写一些应用，用于Linq。
- [ ] Gitbook上怎么索引一个程序，这样就不要占用太多空间了。
- [ ] 自己写一个常见的优化算法的C++程序,涉及最常用的一些算法，主要是拟牛顿法这些，还有一些常见的机器学习算法，可以直接在工业中应用的高质量代码
  - [ ] 要保证库的基础性，不能依赖其它的库，但是也要保证库代码量尽量小

20180509

- [ ] 任何时候，都需要对自己的决策(无论是投资还是工作选择)持怀疑态度，反身思考，以确证自己决策的合理性
- [ ] 怎么通过密码学保证你写的东西的时间是没有被篡改的？
- [x] 把密码学笔记传到Gitbook上，通过图片的形式，有时间的话，要转成电子文字版而不是图像版
- [ ] 35988250+Li-Simon@users.noreply.github.com Li-Simon

20180510

- [ ] Mathjax不能用
- [ ] 有空就思考用blockchain来做一些事情，知识的最终目的还是要转化成产品，变成有价值的东西。学习能力比知识储备更重要，只学不用会花费大量时间，因此更强调在应用的过程中学习。
- [x] 一个类里面实现多组接口
- [x] 整理Bitcoin Code中一些基本类与函数，这个搭建整个BTC的基石，同步到Github上面去。

20180512

- [ ] 第四章第一节最好把不同的类分开介绍
- [x] http://bitcoin-on-nodejs.ebookchain.org/ 这本书要看看

20180513

- [ ] 比特币交易脚本(语言操作符)也就是未来的智能合约，得仔细看
- [ ] 怎么代码区调试智能合约，来明白其过程？因为涉及到交易，可能还是交易链，怎么去调试呢？
- [ ] 在算法介绍之后，最后附上代码，可以直接链接到github上去，这样理论实践结合，更有价值。

20180514

- [x] 既然涉及到了反射，那么就要讲下抽象工厂，进而总结下简单工厂，工厂模式，抽象工厂，进而需要把设计模式变成一章。用到的就总结，必须要附上自己的代码
- [ ] MR中用到的C#技巧要整理下，比如多线程，反射，设计模式，
- [ ] 或许错过了AI，但是不能再错过区块链了，得抓紧。

20180515

- [ ] 兴趣+主动+聪明 比当前的能力更重要10倍。这样的人，在我们团队的成长非常快，因为你可以直接来争取“想做的事情/职位”，公司一定竭尽所能地给你锻炼的机会，只要你能对你自己负责，把事情真正做到位。但如果是“推一下，动一下”的人，我们一定不会要。
- [ ] 自己要在Github上编写自己的程序库。主要包含算法与技术方面的东西
  - [ ] 数据库SQL--设计数据库表格，编写SQL语句
  - [ ] * [ ] 自己的一个项目--线性代数库，主要是算法方面的，包含所有算法工程师，机器学习工程师能用到的算法
  - [ ] 矩阵计算--SVD, PCA, LU, 本征值 这个项目要能体现你有足够的疏离基础
  - [ ] 设置UI，支持数据导入与不同算法的选择
  - [ ] * [ ] 机器学习算法库，常见的机器学习算法，来体现你懂机器学习并能胜任
  - [ ] 分类，聚类，关联，时序，回归
  - [ ] 随机森林
  - [ ] * [ ] 继承Tensorflow并开发一些新功能
- [ ] 每个方向精通一个就可以，其它的有时间了就了解下
  - [ ] 深度学习工具---Tensorflow
  - [ ] 区块链--Bitcoin, 以太坊
  - [ ] 编程语言 C++
- [ ] 最重要的还是C++编程能力(10000行代码)

    - [ ] 通过自己编写的线性代数库来锻炼
    - [ ] 要自己设计，自己写，不要先看别人的，写好之后去参考别人，来修改，升级自己的代码。
- [ ] 代码写好之后要有文档，学会写release note这些东西，还要介绍里面用到的技术以及要写使用文档。

20180516

- [ ] 闪电网络，以及跨链交易会是很重要的的方向
    - [ ] 闪电网络使得小额交易能瞬间完成。
    - [ ] 跨链支付使得不同虚拟币之间能进行交易。
- [ ] 恒星staller也在部署闪电网络 Lightning Labs
- [ ] 要习惯在Linux下进行开发
- [ ] 区块链与（深度学习）算法都要练习，主要通过代码量与基础理论
    - [ ] 区块链--闪电网络
    - [ ] 深度学习--Tensorflow，机器学习算法

20180517

- [ ] 设计线性代数库，LU分解已经写好，接下来可以利用LU分解做
    - [ ] 求线性方程组的解 https://blog.csdn.net/u011584941/article/details/44541127 加一个Vector类
    - [ ] 求矩阵行列式
- [ ] 最小二乘法
    - [ ] SVD
    - [ ] PCA
- [ ] SVD，PCA，线性方程组 求解这些算法封装在一个类里面，而不是CMatrix中，Matrix里面只涉及与matrix属性的东西。
- [ ] 以后会涉及非线性优化的算法
    - [ ] 拟牛顿法、最速下降法、共轭梯度法、信赖域法
    - [ ] 可以先只是函数，最后封装了一个类里面。
- [ ] 多线程问题，自己编写一个多线程例子，不同线程做不同的事情

20180518

- [ ] 把自己练习的project代码放在github上，这样就有一个代码check in的历史纪录，别人也可以看到你在这项目上花费的心血，更重要的是自己能通过代码的迭代来提升自己的实力。

20180519

- [ ] 思考知识图谱的搭建，类比于Google Earth,让学知识像浏览地图一样愉悦。这一切也得靠众人之力。就像Google Earth，如果很多人使用并且在每个地方都拍照上传的话，我们基本上坐在电脑前就可以周游世界的，Google Earth提供了大致的地图框架，而人类上传的照片则是更精细的地图结构。这样两层结构足以让我们对这地区有大致的了解。
- [ ] 知识图谱呢？首先可以基于Wikipedia搭建总体的知识框架，然后依靠众人之力，对细分领域进行补充。

20180521

- [ ] 其实搭建在以太坊上的虚拟货币可以看成是基于以太坊的DAPP,开发难度不大，主要看的是你的想法，因此可以参考这些虚拟货币来设计自己的DAPP.
- [ ] 去中心化的交易平台怎么设计？他是否能实现？
    - [ ] 类似于淘宝的平台
- [ ] 挖矿，因为有奖金去激励矿工，而一些基于p2p的下载utorrent,电驴却没有这些物质激励，因此没有人原因长期打开电脑或者服务器供别人下载资源，如果有物质激励，则会有人购买设备，存储资源来供别人下载资源。
    - [ ] BitTorrent协议能在哪些领域不带来侵权而又能带来收益？
    - [ ] BItTorrent导致的侵权行为

20180522

- [ ] 基于Dotnet(?core)开发一个BTC 界面钱包，
    - [ ] 接下来可能要转到QT上面去开发，
    - [ ] 然后添加不同的虚拟币，向其它钱包学习
- [ ] ProcessProgramMagn.cpp要仔细看

20180523

- [ ] 设计自己的钱包，规划需要哪些功能。一开始就接收，发送 生成比特币地址 接收比特币 发出比特币 生成并导出私钥 对自己的每笔交易，进行签名核实 保护你的资产
- [ ] 借鉴BItcoin Core, Bitpie
- [ ] 先基于NBitcoin,用C#开发，但是得先看看NBitcoin文档,看怎么用。
- [ ] Miura数据存储部分MeasureResultData
- [ ] Test Code

20180531

- [ ] 怎么使用Bitcoin core中的BOOST_AUTO_TEST_CASE
- [ ] 重新Build Bitcoin Linux开发环境并保存一份在移动硬盘(这周)
  - [ ] 只能在家里的网络里面，办公室里面网络不能下载
  - [ ] 写wireshark
- [ ] Socket简单例子还有点问题，怎么让server能一直正常的响应client的请求，就像QQ聊天一样。
  - [ ] QQ运转机制是用户与用户之间交流通过了腾讯的server而进行的，是中心化的聊天方法。
  - [ ] 现在要做去中心化的程序(DAPP),这样就是去掉了腾讯的中心服务器角色，使得每个每个人即 client又是server,数据可以由自己保存，或者通过矿工提供数据保存服务。在这里，矿工的角色就是数据保存，但是数据是十分庞大的，得想着把它分散到许多网络节点上，这样分散也有一个好处就是防止数据垄断。
  - [ ] 防止数据篡改可以由加密算法来实现。交流双方是否可以通过2-2合约来实现数据的加密，只有两人再次聊天时，数据才能被打开。
  - [ ] 这样可以保证数据的安全性。但是怎么盈利呢？这这个网络里，交流不是免费的，得有gas去驱动，这样才有矿工来提供数据保存服务。

201806

20180601

- [ ] 生物医学成像算法(两个主流的)看仔细，并看看实现

20180602

- [ ] NBXplorer的使用，以及用来做钱包
- [ ] 再看看LTC,DASH这些，思考为啥他们可以合在一个框架下？是否可以添加其它的数字货币？

20180604

- [ ] 图像处理与机器视觉 VC++与matlab版
  - [ ] 看看主要介绍的理论与算法，思考怎么用C++实现。能实现才是最重要的，不能实现，空谈理论没啥价值。
- [ ] CT成像算法
- [ ] 整理自己机器学习方面的笔记

20180605

- [ ] 同步与内核对象要熟练使用

第四章 区块链

第一节 比特币代码分析

Validation.cpp
BTC 基本类

https://en.bitcoin.it/wiki/Dump_format

https://en.bitcoin.it/wiki/Bitcoin_Core_0.11_(ch_6):_The_Blockchain

https://bitcointalk.to/index.php?topic=1641447.0

```
class COutPoint
{
public:
    uint256 hash;  //
    uint32_t n;
}
COutPoint(000000, -1)
Part of a CTxIn structure, this references a specific output. The first value is the
truncated hash of the transaction, and the second value is an output index.
```

```
class CTxIn
{
public:
    COutPoint prevout;
    CScript scriptSig;
    uint32_t nSequence;
    CScriptWitness scriptWitness; //! Only serialized through CTransaction
}
CTxIn(COutPoint(237fe8, 0), scriptSig=0xA87C02384E1F184B79C6AC)
CTxIn(COutPoint(000000, -1), coinbase 04ffff001d010)
CTxIn(COutPoint(237fe8, 0), scriptSig=0xA87C02384E1F184B79C6AC, nSequence=5000)
A transaction input.
scriptSig is the truncated scriptSig of the input. "coinbase" is the non-truncated scriptSig of a generation input.
nSequence appears only if the input has a non-default sequence.
Sequence numbers are intended for use with the transaction replacement feature, but can currently be safely ignored.
```

```
class CTxOut
{
public:
    CAmount nValue;
    CScript scriptPubKey;
}
CTxOut(nValue=50.00000000, scriptPubKey=OP_DUP OP_HASH160 0x1512)
A transaction output.
nValue is the the full-precision output value.
scriptPubKey is the truncated CScript for the output (the entire script string is truncated to 30 characters).
```

```
class CTransaction
{
public:
    const int32_t nVersion;
    const std::vector<CTxIn> vin;
    const std::vector<CTxOut> vout;
    const uint32_t nLockTime;
private:
    /** Memory only. */
    const uint256 hash;
}
CTransaction(hash=4a5e1e, ver=1, vin.size=1, vout.size=1, nLockTime=0)
CTxIn(COutPoint(000000, -1), coinbase 04ffff001d010)
CTxOut(nValue=50.00000000, scriptPubKey=0x5F1DF16B2B704C8A578D0B)
A transaction. hash is the truncated hash. ver is the transaction version.
vout.size is the number of outputs.
nLockTime is intended for use with transaction replacement, and is not currently used for anything useful.
```

```
class CBlockHeader
{
public:
    // header
    int32_t nVersion;
    uint256 hashPrevBlock;
```

```
    uint256 hashMerkleRoot;
    uint32_t nTime;
    uint32_t nBits;
    uint32_t nNonce;
}
class CBlock : public CBlockHeader
{
public:
    // network and disk
    std::vector<CTransactionRef> vtx;
}
CBlock(hash=00000000009ffdadbb2a, ver=1, hashPrevBlock=0000000000b079382c19, hashMerkleRoot=e81287,
nTime=1281156783, nBits=1c00ba18, nNonce=2283211008, vtx=6)
  CTransaction(hash=2d7f4d, ver=1, vin.size=1, vout.size=1, nLockTime=0)
    CTxIn(COutPoint(000000, -1), coinbase 0418ba001c02ce03)
    CTxOut(nValue=50.00000000, scriptPubKey=0x4FE11D72F988AEA611F026)
  CTransaction(hash=3407a8, ver=1, vin.size=2, vout.size=1, nLockTime=0)
    CTxIn(COutPoint(df39bf, 0), scriptSig=0x01DD1AD8E9EFE65B70E983)
    CTxIn(COutPoint(64ebea, 1), scriptSig=0x0165A1F9873BA16265D9C4)
    CTxOut(nValue=0.06000000, scriptPubKey=OP_DUP OP_HASH160 0xEDEF)
  CTransaction(hash=5edf5a, ver=1, vin.size=1, vout.size=1, nLockTime=0)
    CTxIn(COutPoint(b77e0f, 0), scriptSig=0x01E39C53AFC1B9BE02E53A)
    CTxOut(nValue=350.00000000, scriptPubKey=OP_DUP OP_HASH160 0xD7EF)
  CTransaction(hash=65c356, ver=1, vin.size=1, vout.size=2, nLockTime=0)
    CTxIn(COutPoint(893335, 0), scriptSig=0x01B8C315FD58F0DFA0DEA2)
    CTxOut(nValue=1.85850000, scriptPubKey=OP_DUP OP_HASH160 0x3181)
    CTxOut(nValue=3.14150000, scriptPubKey=OP_DUP OP_HASH160 0xF99E)
  CTransaction(hash=89aa32, ver=1, vin.size=1, vout.size=2, nLockTime=0)
    CTxIn(COutPoint(4a7469, 0), scriptSig=0x010D15199DCE4B11D391CF)
    CTxOut(nValue=0.05000000, scriptPubKey=OP_DUP OP_HASH160 0x5603)
    CTxOut(nValue=0.20000000, scriptPubKey=OP_DUP OP_HASH160 0x6074)
  CTransaction(hash=e3e69c, ver=1, vin.size=1, vout.size=2, nLockTime=0)
    CTxIn(COutPoint(b77e0f, 1), scriptSig=0x012E18AF1264180255E0C3)
    CTxOut(nValue=50.00000000, scriptPubKey=OP_DUP OP_HASH160 0x458E)
    CTxOut(nValue=100.00000000, scriptPubKey=OP_DUP OP_HASH160 0x9EEF)
  vMerkleTree: 2d7f4d 3407a8 5edf5a 65c356 89aa32 e3e69c 8ebc6a d5e414 89b77c d1074c 70a4e6 e81287

  A block. Hash is the truncated hash. Version is the block version.
  HashPrevBlock is the truncated hash of the previous block.
  HashMerkleRoot is the truncated Merkle root. nTime is the Unix timestamp of the block.
  nBits is the current target in compact format. nNonce is the nonce.
  vtx is the number of transactions.
```

```
class CChain {
private:
    std::vector<CBlockIndex*> vChain;
}
```

```
/** The block chain is a tree shaped structure starting with the
 * genesis block at the root, with each block potentially having multiple
 * candidates to be the next block. A blockindex may have multiple pprev pointing
 * to it, but at most one of them can be part of the currently active branch.
class CBlockIndex
{
public:
    //! pointer to the hash of the block, if any. Memory is owned by this CBlockIndex
    const uint256* phashBlock;

    //! pointer to the index of the predecessor of this block
    CBlockIndex* pprev;

    //! pointer to the index of some further predecessor of this block
    CBlockIndex* pskip;

    //! height of the entry in the chain. The genesis block has height 0
    int nHeight;

    //! Which # file this block is stored in (blk?????.dat)
    int nFile;

    //! Byte offset within blk?????.dat where this block's data is stored
    unsigned int nDataPos;

    //! Byte offset within rev?????.dat where this block's undo data is stored
    unsigned int nUndoPos;

    //! (memory only) Total amount of work (expected number of hashes) in the chain up to and including this block
    arith_uint256 nChainWork;

    //! Number of transactions in this block.
```

```
    //! Note: in a potential headers-first mode, this number cannot be relied upon
    unsigned int nTx;

    //! (memory only) Number of transactions in the chain up to and including this block.
    //! This value will be non-zero only if and only if transactions for this block and all its parents are available.
    //! Change to 64-bit type when necessary; won't happen before 2030
    unsigned int nChainTx;

    //! Verification status of this block. See enum BlockStatus
    unsigned int nStatus;

    //! block header
    int nVersion;
    uint256 hashMerkleRoot;
    unsigned int nTime;
    unsigned int nBits;
    unsigned int nNonce;

    //! (memory only) Sequential id assigned to distinguish order in which blocks are received.
    int32_t nSequenceId;

    //! (memory only) Maximum nTime in the chain upto and including this block.
    unsigned int nTimeMax;
}
```

```
class CNodeStats
{
public:
    NodeId nodeid;
    ServiceFlags nServices;
    bool fRelayTxes;
    int64_t nLastSend;
    int64_t nLastRecv;
    int64_t nTimeConnected;
    int64_t nTimeOffset;
    std::string addrName;
    int nVersion;
    std::string cleanSubVer;
    bool fInbound;
    bool fAddnode;
    int nStartingHeight;
    uint64_t nSendBytes;
    mapMsgCmdSize mapSendBytesPerMsgCmd;
    uint64_t nRecvBytes;
    mapMsgCmdSize mapRecvBytesPerMsgCmd;
    bool fWhitelisted;
    double dPingTime;
    double dPingWait;
    double dMinPing;
    std::string addrLocal;
    CAddress addr;
};
```

```
/**
 * A CWallet is an extension of a keystore, which also maintains a set of transactions and balances,
 * and provides the ability to create new transactions.
 */
class CWallet : public CCryptoKeyStore, public CValidationInterface
{
private:
    static std::atomic<bool> fFlushThreadRunning;
    CWalletDB *m_pwalletdbEncryption;

    //! the current wallet version: clients below this version are not able to load the wallet
    int m_nWalletVersion;

    //! the maximum wallet format version: memory-only variable that specifies to what version this wallet may be upgraded
    int m_nWalletMaxVersion;

    int64_t nNextResend;
    int64_t nLastResend;
    bool fBroadcastTransactions;
    TxSpends mapTxSpends;
    void AddToSpends(const COutPoint& outpoint, const uint256& wtxid);
    void AddToSpends(const uint256& wtxid);

    /* Mark a transaction (and its in-wallet descendants) as conflicting with a particular block. */
    void MarkConflicted(const uint256& hashBlock, const uint256& hashTx);

    void SyncMetaData(std::pair<TxSpends::iterator, TxSpends::iterator>);
```

```
    /* the HD chain data model (external chain counters) */
    CHDChain m_hdChain;

    bool m_fFileBacked;

    std::set<int64_t> setKeyPool;
    static CFeeRate minTxFee;
    static CFeeRate fallbackFee;

    void SetNull()
    {
        m_nWalletVersion = FEATURE_BASE;
        m_nWalletMaxVersion = FEATURE_BASE;
        m_fFileBacked = false;
        nMasterKeyMaxID = 0;
        m_pwalletdbEncryption = NULL;
        nOrderPosNext = 0;
        nNextResend = 0;
        nLastResend = 0;
        nTimeFirstKey = 0;
        fBroadcastTransactions = false;
    }
}
```

```
class CValidationInterface {
protected:
    virtual void UpdatedBlockTip(const CBlockIndex *pindexNew, const CBlockIndex *pindexFork, bool fInitialDownload) {}
    virtual void SyncTransaction(const CTransaction &tx, const CBlockIndex *pindex, int posInBlock) {}
    virtual void SetBestChain(const CBlockLocator &locator) {}
    virtual void UpdatedTransaction(const uint256 &hash) {}
    virtual void Inventory(const uint256 &hash) {}
    virtual void ResendWalletTransactions(int64_t nBestBlockTime, CConnman* connman) {}
    virtual void BlockChecked(const CBlock&, const CValidationState&) {}
    virtual void GetScriptForMining(boost::shared_ptr<CReserveScript>&) {};
    virtual void ResetRequestCount(const uint256 &hash) {};
    virtual void NewPoWValidBlock(const CBlockIndex *pindex, const std::shared_ptr<const CBlock>& block) {};
    friend void ::RegisterValidationInterface(CValidationInterface*);
    friend void ::UnregisterValidationInterface(CValidationInterface*);
    friend void ::UnregisterAllValidationInterfaces();
};
```

```
/** Keystore which keeps the private keys encrypted.
 * It derives from the basic key store, which is used if no encryption is active.
 */
class CCryptoKeyStore : public CBasicKeyStore
{
private:
    CryptedKeyMap mapCryptedKeys;

    CKeyingMaterial vMasterKey;

    //! if fUseCrypto is true, mapKeys must be empty
    //! if fUseCrypto is false, vMasterKey must be empty
    bool fUseCrypto;

    //! keeps track of whether Unlock has run a thorough check before
    bool fDecryptionThoroughlyChecked;

protected:
    bool SetCrypted();

    //! will encrypt previously unencrypted keys
    bool EncryptKeys(CKeyingMaterial& vMasterKeyIn);

    bool Unlock(const CKeyingMaterial& vMasterKeyIn);
}
```

```
/** Basic key store, that keeps keys in an address->secret map */
class CBasicKeyStore : public CKeyStore
{
protected:
    KeyMap mapKeys;
    WatchKeyMap mapWatchKeys;
    ScriptMap mapScripts;
    WatchOnlySet setWatchOnly;
}
```

```
/** A virtual base class for key stores */
class CKeyStore
{
protected:
    mutable CCriticalSection cs_KeyStore;

public:
    virtual ~CKeyStore() {}

    //! Add a key to the store.
    virtual bool AddKeyPubKey(const CKey &key, const CPubKey &pubkey) =0;
    virtual bool AddKey(const CKey &key);

    //! Check whether a key corresponding to a given address is present in the store.
    virtual bool HaveKey(const CKeyID &address) const =0;
    virtual bool GetKey(const CKeyID &address, CKey& keyOut) const =0;
    virtual void GetKeys(std::set<CKeyID> &setAddress) const =0;
    virtual bool GetPubKey(const CKeyID &address, CPubKey& vchPubKeyOut) const =0;

    //! Support for BIP 0013 : see https://github.com/bitcoin/bips/blob/master/bip-0013.mediawiki
    virtual bool AddCScript(const CScript& redeemScript) =0;
    virtual bool HaveCScript(const CScriptID &hash) const =0;
    virtual bool GetCScript(const CScriptID &hash, CScript& redeemScriptOut) const =0;

    //! Support for Watch-only addresses
    virtual bool AddWatchOnly(const CScript &dest) =0;
    virtual bool RemoveWatchOnly(const CScript &dest) =0;
    virtual bool HaveWatchOnly(const CScript &dest) const =0;
    virtual bool HaveWatchOnly() const =0;
};
```

```
class CRPCCommand
{
public:
    std::string category;
    std::string name;
    rpcfn_type actor;
    bool okSafeMode;
    std::vector<std::string> argNames;
};
```

```
/**
 * Bitcoin RPC command dispatcher.
 */
class CRPCTable
{
private:
    std::map<std::string, const CRPCCommand*> mapCommands;
public:
    CRPCTable();
    const CRPCCommand* operator[](const std::string& name) const;
    std::string help(const std::string& name) const;

    /**
     * Execute a method.
     * @param request The JSONRPCRequest to execute
     * @returns Result of the call.
     * @throws an exception (UniValue) when an error happens.
     */
    UniValue execute(const JSONRPCRequest &request) const;

    /**
    * Returns a list of registered commands
    * @returns List of registered commands.
    */
    std::vector<std::string> listCommands() const;


    /**
     * Appends a CRPCCommand to the dispatch table.
     * Returns false if RPC server is already running (dump concurrency protection).
     * Commands cannot be overwritten (returns false).
     */
    bool appendCommand(const std::string& name, const CRPCCommand* pcmd);
};
```

```
/** A reference to a CKey: the Hash160 of its serialized public key */
class CKeyID : public uint160
```

```
{
public:
    CKeyID() : uint160() {}
    CKeyID(const uint160& in) : uint160(in) {}

public:
    base58string GetBase58addressWithNetworkPubkeyPrefix() const;
};
```

```
/** A reference to a CKey: the Hash160 of its serialized public key */
class CKeyID : public uint160
{
public:
    CKeyID() : uint160() {}
    CKeyID(const uint160& in) : uint160(in) {}

public:
    base58string GetBase58addressWithNetworkPubkeyPrefix() const;
};
```

```
/** An encapsulated public key. */
class CPubKey
{
private:

    /**
     * Just store the serialized data.
     * Its length can very cheaply be computed from the first byte.
     */
    unsigned char m_vch[65];
}
```

```
/** In-flight HTTP request.
 * Thin C++ wrapper around evhttp_request.
 */
class HTTPRequest
{
private:
    struct evhttp_request* req;
    bool replySent;

public:
    HTTPRequest(struct evhttp_request* req);
    ~HTTPRequest();

    enum RequestMethod {
        UNKNOWN,
        GET,
        POST,
        HEAD,
        PUT
    };
}
```

```
struct CBlockTemplate
{
    CBlock block;
    std::vector<CAmount> vTxFees;
    std::vector<int64_t> vTxSigOpsCost;
    std::vector<unsigned char> vchCoinbaseCommitment;
};
```

```
/** A transaction with a merkle branch linking it to the block chain. */
class CMerkleTx
{
private:
    /** Constant used in hashBlock to indicate tx has been abandoned */
    static const uint256 ABANDON_HASH;

public:
    CTransactionRef tx;
    uint256 hashBlock;

    /* An nIndex == -1 means that hashBlock (in nonzero) refers to the earliest
    * block in the chain we know this or any in-wallet dependency conflicts
```

```
     * with. Older clients interpret nIndex == -1 as unconfirmed for backward
     * compatibility.
     */
    int nIndex;
}
```

第二节 Bitcoin-cli API

```
A、一般性的命令
help ( "command" )
stop
getinfo
ping
getnettotals
getnetworkinfo
getpeerinfo
getconnectioncount
verifychain ( checklevel numblocks )
getaddednodeinfo dns ( "node" )
addnode "node" "add|remove|onetry"



B、钱包、账户、地址、转帐、发消息
getwalletinfo
walletpassphrase "passphrase" timeout
walletlock
walletpassphrasechange "oldpassphrase" "newpassphrase"
backupwallet "destination"
importwallet "filename"
dumpwallet "filename"

listaccounts ( minconf )
getaddressesbyaccount "account"
getaccountaddress "account"
getaccount "bitcoinaddress"
validateaddress "bitcoinaddress"
dumpprivkey "bitcoinaddress"
setaccount "bitcoinaddress" "account"
getnewaddress ( "account" )
keypoolrefill ( newsize )
importprivkey "bitcoinprivkey" ( "label" rescan )
createmultisig nrequired ["key",...]
addmultisigaddress nrequired ["key",...] ( "account" )

getbalance ( "account" minconf )
getunconfirmedbalance
getreceivedbyaccount "account" ( minconf )
listreceivedbyaccount ( minconf includeempty )
getreceivedbyaddress "bitcoinaddress" ( minconf )
listreceivedbyaddress ( minconf includeempty )
move "fromaccount" "toaccount" amount ( minconf "comment" )
listunspent ( minconf maxconf  ["address",...] )
listlockunspent
lockunspent unlock [{"txid":"txid","vout":n},...]


getrawchangeaddress
listaddressgroupings
settxfee amount
sendtoaddress "bitcoinaddress" amount ( "comment" "comment-to" )
sendfrom "fromaccount" "tobitcoinaddress" amount ( minconf "comment" "comment-to" )
sendmany "fromaccount" {"address":amount,...} ( minconf "comment" )

signmessage "bitcoinaddress" "message"
verifymessage "bitcoinaddress" "signature" "message"


C、Tx、Block、Ming

createrawtransaction [{"txid":"id","vout":n},...] {"address":amount,...}
signrawtransaction "hexstring" ( [{"txid":"id","vout":n,"scriptPubKey":"hex","redeemScript":"hex"},...] ["privatekey1",...] sighashtype )
sendrawtransaction "hexstring" ( allowhighfees )


gettransaction "txid"
listtransactions ( "account" count from )
listsinceblock ( "blockhash" target-confirmations )


getrawmempool ( verbose )
gettxoutsetinfo
gettxout "txid" n ( includemempool )
getrawtransaction "txid" ( verbose )
decoderawtransaction "hexstring"
```

```
decodescript "hex"

getblockchaininfo
getblockcount
getbestblockhash
getblockhash index
getblock "hash" ( verbose )

getmininginfo
getdifficulty
getnetworkhashps ( blocks height )
gethashespersec

getgenerate
setgenerate generate ( genproclimit )
getwork ( "data" )
getblocktemplate ( "jsonrequestobject" )
submitblock "hexdata" ( "jsonparametersobject" )
```

比特币客户端bitcoind的高级用法

Bitcoin 比特币官方客户端有两个版本：一个是图形界面的版本，通常被称为 Bitcoin（首字母大写），以及一个简洁命令行的版本（称为 bitcoind）。它们相互间是兼容的，有着同样的命令行参数，读取相同的配置文件，也读写相同的数据文件。您可以在一台电脑中运行 Bitcoin 客户端或是 bitcoind 客户端的其中一个（如果您不小心尝试同时运行另外一个客户端，它会提示您已经有一个客户端在运行并且自动退出）。

命令行参数
使用 -? 或 –help 参数运行 Bitcoin 或 bitcoind，它会提示常用的命令行参数并退出。

用法：

bitcoind [选项]
bitcoind [选项] <命令> [参数] 将命令发送到 -server 或 bitcoind
bitcoind [选项] help 列出命令
bitcoind [选项] help <命令> 获取该命令的帮助

选项：

-conf=<文件名> 指定配置文件（默认：bitcoin.conf）
-pid=<文件名> 指定 pid（进程 ID）文件（默认：bitcoind.pid）
-gen 生成比特币
-gen=0 不生成比特币
-min 启动时最小化
-splash 启动时显示启动屏幕（默认：1）
-datadir=<目录名> 指定数据目录
-dbcache= 设置数据库缓存大小，单位为兆字节（MB）（默认：25）
-dblogsize= 设置数据库磁盘日志大小，单位为兆字节（MB）（默认：100）
-timeout= 设置连接超时，单位为毫秒
-proxy= 通过 Socks4 代理链接
-dns addnode 允许查询 DNS 并连接
-port=<端口> 监听 <端口> 上的连接（默认：8333，测试网络 testnet：18333）
-maxconnections= 最多维护 个节点连接（默认：125）
-addnode= 添加一个节点以供连接，并尝试保持与该节点的连接
-connect= 仅连接到这里指定的节点
-irc 使用 IRC（因特网中继聊天）查找节点（默认：0）
-listen 接受来自外部的连接（默认：1）
-dnsseed 使用 DNS 查找节点（默认：1）
-banscore= 与行为异常节点断开连接的临界值（默认：100）
-bantime= 重新允许行为异常节点连接所间隔的秒数（默认：86400）
-maxreceivebuffer= 最大每连接接收缓存，_1000 字节（默认：10000）
-maxsendbuffer= 最大每连接发送缓存，_1000 字节（默认：10000）
-upnp 使用全局即插即用（UPNP）映射监听端口（默认：0）
-detachdb 分离货币块和地址数据库。会增加客户端关闭时间（默认：0）
-paytxfee= 您发送的交易每 KB 字节的手续费
-testnet 使用测试网络
-debug 输出额外的调试信息
-logtimestamps 调试信息前添加时间戳
-printtoconsole 发送跟踪/调试信息到控制台而不是 debug.log 文件
-printtodebugger 发送跟踪/调试信息到调试器
-rpcuser=<用户名> JSON-RPC 连接使用的用户名
-rpcpassword=<密码> JSON-RPC 连接使用的密码
-rpcport= JSON-RPC 连接所监听的 <端口>（默认：8332）
-rpcallowip= 允许来自指定 地址的 JSON-RPC 连接
-rpcconnect= 发送命令到运行在 地址的节点（默认：127.0.0.1）
-blocknotify=<命令> 当最好的货币块改变时执行命令（命令中的 %s 会被替换为货币块哈希值）
-upgradewallet 将钱包升级到最新的格式
-keypool= 将密匙池的尺寸设置为（默认：100）
-rescan 重新扫描货币块链以查找钱包丢失的交易
-checkblocks= 启动时检查多少货币块（默认：2500，0 表示全部）
-checklevel= 货币块验证的级别（0-6，默认：1）

SSL 选项：

-rpcssl 使用 OpenSSL（https）JSON-RPC 连接
-rpcsslcertificatechainfile=<文件.cert> 服务器证书文件（默认：server.cert）
-rpcsslprivatekeyfile=<文件.pem> 服务器私匙文件（默认：server.pem）
-rpcsslciphers=<密码> 可接受的密码（默认：TLSv1+HIGH:!SSLv2:!aNULL:!eNULL:!AH:!3DES:@STRENGTH）

bitcoin.conf 配置文件

除了 -datadir 和 -conf 以外的所有命令行参数都可以通过一个配置文件来设置，而所有配置文件中的选项也都可以在命令行中设置。命令行参数设置的值会覆盖配置文件中的设置。

配置文件是“设置=值”格式的一个列表，每行一个。您还可以使用 # 符号来编写注释。

配置文件不会自动创建；您可以使用您喜爱的纯文本编辑器来创建它。默认情况下，Bitcoin（或 bitcoind）会在比特币数据文件夹下查找一个名为“bitcoin.conf”的文件，但是数据文件夹和配置文件的路径都可以分别通过 -datadir 和 -conf 命令行参数分别指定。

操作系统

默认数据文件夹

配置文件路径

Windows

%APPDATA%\Bitcoin\

(XP) C:\Documents and Settings\username\Application Data\Bitcoin\bitcoin.conf

(Vista, 7) C:\Users\username\AppData\Roaming\Bitcoin\bitcoin.conf

Linux

$HOME/.bitcoin/

/home/username/.bitcoin/bitcoin.conf

Mac OSX

$HOME/Library/Application Support/Bitcoin/

/Users/username/Library/Application Support/Bitcoin/bitcoin.conf

注意：如果 Bitcoin 比特币客户端测试网模式运行，在数据文件夹下客户端会自动创建名为“testnet”的子文件夹。
bitcoin.conf 示例

bitcoin.conf 配置文件。以 # 开头的行是注释。

网络相关的设置：

在测试网络中运行，而不是在真正的比特币网络

testnet=0

通过一个 Socks4 代理服务器连接

proxy=127.0.0.1:9050

addnode 与 connect 的区别

假设您使用了 addnode=4.2.2.4 参数，那么 addnode 便会与您的节点连接，并且告知您的节点所有与它相连接的其它节点。另外它还会将您的节点信息告知与其相连接的其它节点，这样它们也可以连接到您的节点。connect 在您的节点“连接”到它的时候并不会做上述工作。仅它会与您连接，而其它节点不会。因此如果您位于防火墙后，或者因为其它原因无法找到节点，则使用“addnode”添加一些节点。如果您想保证隐私，使用“connect”连接到那些您可以“信任”的节点。如果您在一个局域网内运行了多个节点，您不需要让它们建立许多 连接。您只需要使用“connect”让它们统一连接到一个已端口转 发并拥有多个连接的节点。您可以在下面使用多个 addnode= 设置来连接到指定的节点 addnode=69.164.218.197 addnode=10.0.0.2:8333 ... 或使用多个 connect= 设置来仅连接到指定的节点 connect=69.164.218.197 connect=10.0.0.1:8333 不使用因特网中继聊天（IRC）（irc.lfnet.org #bitcoin 频道） 来查找其它节点 noirc=0 入站+出站的最大连接数 maxconnections= JSON-RPC 选项（用于控制运行中的 Bitcoin/bitcoind 进程）：server=1 告知 Bitcoin-QT 接受 JSON-RPC 命令 server=0 您必须设置 rpcuser 和 rpcpassword 以确保 JSON-RPC 的安全 rpcuser=Ulysseys rpcpassword=YourSuperGreatPasswordNumber_DO_NOT_USE_THIS_OR_YOU_WILL_GET_ROBBED_385593 客户端在 HTTP 连接建立后，等待多少秒以完成一个 RPC HTTP 请求 rpctimeout=30 默认仅允许来自本机的 RPC 连接。在这里您可以指定多个 rpcallowip=，来设置您想允许连接的其它主机 IP 地址。您可以使用 * 作为通配符。rpcallowip=10.1.1.34

rpcallowip=192.168.1.*

在如下端口监听 RPC 连接

rpcport=8332

您可以通过如下设置使用 Bitcoin 或 bitcoind 来发送命令到一个在

其它主机远程运行的 Bitcoin/bitcoind 客户端

rpcconnect=127.0.0.1

使用安全套接层(也称为 TLS 或 HTTPS)来

连接到 Bitcoin -server 或 bitcoind

rpcssl=1

当 rpcssl=1 时使用的 OpenSSL 设置

rpcsslciphers=TLSv1+HIGH:!SSLv2:!aNULL:!eNULL:!AH:!3DES:@STRENGTH

rpcsslcertificatechainfile=server.cert

rpcsslprivatekeyfile=server.pem

其它选项:

设置 gen=1 以尝试生成比特币(采矿)

gen=0

预生成如下数目的公匙和私匙, 这样钱包备份便可以对已有的交易以及未来

多笔交易有效

keypool=100

每次您发送比特币的时候支付一个可选的额外的交易手续费。包含手续费的交易

会更快的被包含在新生成的货币块中, 因此会更快生效

paytxfee=0.00

允许直接连接, 实现"通过 IP 地址支付"功能

allowreceivebyip=1

用户界面选项:

最小化启动比特币客户端

min=1

最小化到系统托盘

minimizetotray=1

基本知识

地址

钱包由私钥与公钥(地址)组成，通过私钥可以得到公钥，这是由密码学保证的，类似于

$Y = C^X$ mod D 计算都是模掉一个大数D

其中C是已知的大的质数，X就是私钥，Y就是公钥，通过私钥X得到公钥Y很简单，但是通过公钥Y求解私钥X则很难

1:创建私钥

```
Key privateKey = new Key(); // generate a random private key
```

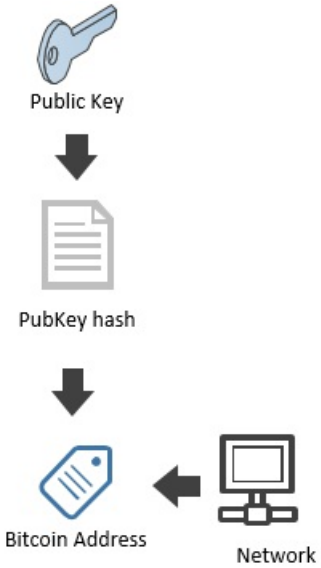
2:获得公钥

```
PubKey publicKey = privateKey.PubKey; //求得公钥
Console.WriteLine(publicKey); // 0251036303164f6c458e9f7abecb4e55e5ce9ec2b2f1d06d633c9653a07976560c
```

3:生成地址:miannet, testnet, regtest上的地址不同

RIPEMD160(SHA256(pubkey))

```
Console.WriteLine(publicKey.GetAddress(Network.Main)); // 1PUYsjwfNmX64wS368ZR5FMouTtUmvtmTY
Console.WriteLine(publicKey.GetAddress(Network.TestNet)); // n3zWAo2eBnxLr3ueohXnuAa8mTVBhxmPhq
```



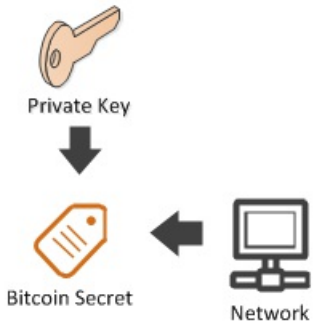
BTC的地址是对公钥做hash之后的结果。

A public key hash is generated by using a SHA256 hash on the public key, then a RIPEMD160 hash on the result, using Big

Endian notation. The function could look like this: RIPEMD160(SHA256(pubkey))

Bitcoin Secret(WIF: Wallet Import Formator)

私钥常表示成Base58Check的形式，称为Bitcoin Secret(also known as Wallet Import Format or simply WIF)

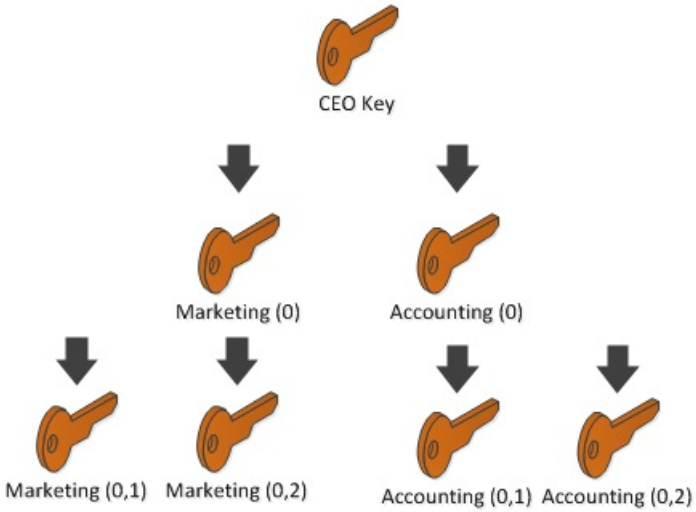



```
Key privateKey = new Key(); // generate a random private key
BitcoinSecret mainNetPrivateKey = privateKey.GetBitcoinSecret(Network.Main); // generate our Bitcoin secret(also known as Wallet Import Format or simply WIF) from our private key for the mainnet
BitcoinSecret testNetPrivateKey = privateKey.GetBitcoinSecret(Network.TestNet); // generate our Bitcoin secret(also known as Wallet Import Format or simply WIF) from our private key for the testnet
Console.WriteLine(mainNetPrivateKey); // L5B67zvrndS5c71EjkrTJZ99UaoVbMUAK58GKdQUfYCpAa6jypvn
Console.WriteLine(testNetPrivateKey); // cVY5auviDh8LmYUW8AfafseD6p6uFoZrP7GjS3rzAerpRKE9Wmuz
bool WifIsBitcoinSecret = mainNetPrivateKey == privateKey.GetWif(Network.Main);
Console.WriteLine(WifIsBitcoinSecret); // True
```

从BitcoinSecret可以得到私钥privateKey,但是不可以从Bitcoin Address得到Public Key，因为Bitcoin Address是Public Key的Hash值。

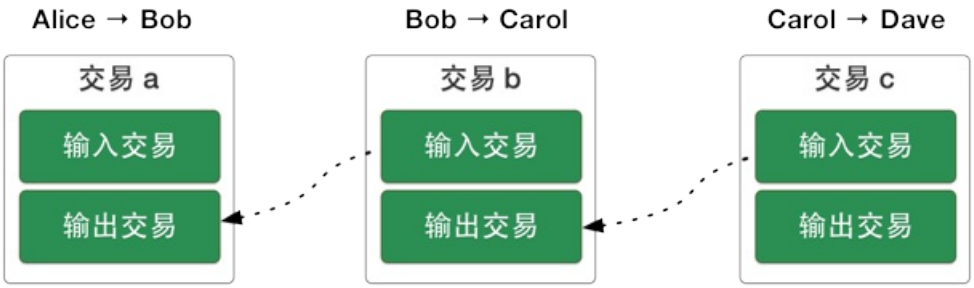
```
QBitNinjaClient client = new QBitNinjaClient(Network.Main);
var transactionID = uint256.Parse("f13dc48fb035bbf0a6e989a26b3ecb57b84f85e0836e777d6edf60d87a4a2d94");
QBitNinja.Client.Models.GetTransactionResponse transactionResponse = client.GetTransaction(transactionID).Result;

NBitcoin.Transaction transaction = transactionResponse.Transaction;
Console.WriteLine("transactionResponse: " + transactionResponse.TransactionId);
Console.WriteLine("transaction Hash: "+transaction.GetHash());
Console.WriteLine("Total out: " + transaction.TotalOut.ToString()); //交易总量

IEnumerable<IndexedTxIn> txIn = transaction.Inputs.AsIndexedInputs();
foreach(IndexedTxIn trx in txIn)
    Console.WriteLine("\n coin:"+ trx.ScriptSig.ToString());

IEnumerable<Coin> coins = transaction.Outputs.AsCoins();
foreach(Coin coin in coins)
{
    Console.WriteLine(coin.ToString());
    Console.WriteLine();
    var paymentScript = coin.TxOut.ScriptPubKey;
    Console.WriteLine("paymentScript: "+ paymentScript);//每笔输出都有签名，能解密这签名的人可以使用这笔BTC
}
```

BIP44 hierarchy Deterministic Wallet(HD)



第三节 比特币的交易

输入脚本也称签名脚本，输出脚本也称赎回脚本
在这里我们先讨论单输入单输出的比特币交易，因为这样描述起来更方便且不影响对『脚本』的理解。[1]
9c50cee8d50e273100987bb12ec46208cb04a1d5b68c9bea84fd4a04854b5eb1这是一个单输入单输出交易，看下我们要关注的数据：
假设有如下一系列交易

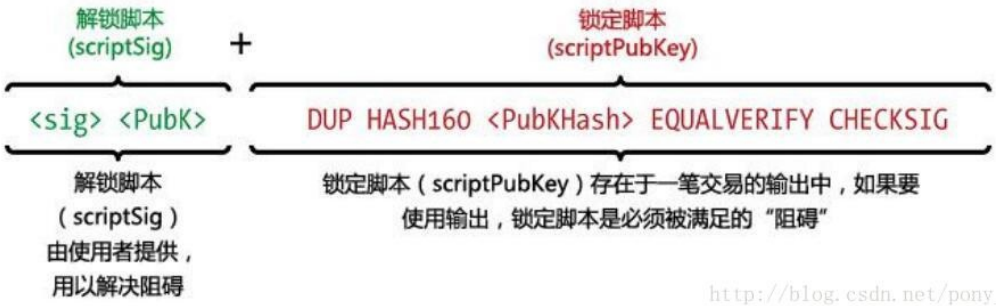


在交易b中，Bob要转比特币给Carol,假设他用的就是Alice转给他的比特币。那么在交易b中，假设输入b脚本如下

> **Hash:**
>
> 9c50cee8d50e273100987bb12ec46208cb04a1d5b68c9bea84fd4a04854b5eb1
>
> 输入交易:
>
> 前导输入的Hash:
> 437b95ae15f87c7a8ab4f51db5d3c877b972ef92f26fbc6d3c4663d1bc750149
>
> 输入脚本 scriptSig(公钥与签名):
> 3045022100efe12e2584bbd346bccfe67fd50a54191e4f45f945e3853658284358d9c062ad02200121e00b6297c0874650d00b786971f5b4601e32b3f81afa9f9f8108e93c752201
> 038b29d4fbbd12619d45c84c83cb4330337ab1b1a3737250f29cec679d7551148a
>
> 输出交易(交易a的输出脚本):
>
> 转账值:
> 0.05010000 btc
>
> 输出脚本 scriptPubKey:
> OP_DUP OP_HASH160 be10f0a78f5ac63e8746f7f2e62a5663eed05788 OP_EQUALVERIFY OP_CHECKSIG

转账值:转移比特币数量

输出脚本scriptPubKey:也就是Bob想用这笔UTXO时需要解锁的脚本，也就是需要Bob去求解的一个方程，这个方程操作如下，以交易b的输入脚本为输入 x，以交易a的输出脚本作为函数 f，最终看能否f(x)是否成立
如下图所示[2]:



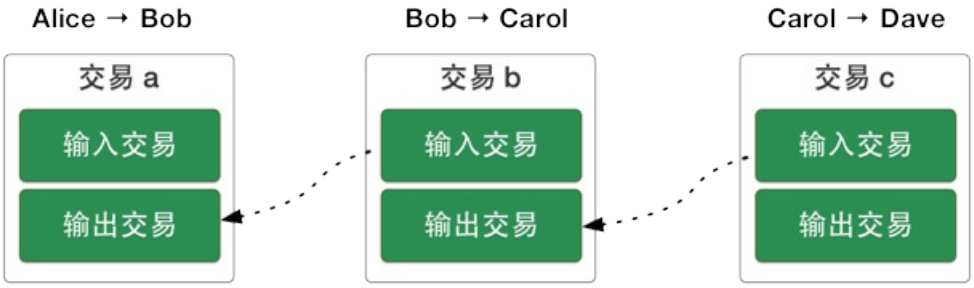
1. OP_DUP ：duplicate operator,复制操作，复制了交易b中PUSHDATA,
2. OP_HASH160:先SHA256再ripemd160得到160位哈希值
3. OP_EQUALVERIFY 检验是否相等
4. OP_CHECKSIG 签名

在Bitcoin Wiki中提到:

> 原先发送币的一方，控制脚本运行，以便比特币在下一个交易中使用。想花掉币的另一方必须把以前记录的运行为真的脚本，放到输入区。

换句话说，在一个交易中，『输出脚本』是数学题，『输入脚本』是题解，但不是这道数学题的题解。我开始看Wiki的时候，在这里遇到了一些障碍，没法理解『输入脚本』和『输出脚本』的联系。但是在考虑交易间的关系后，就明白了。

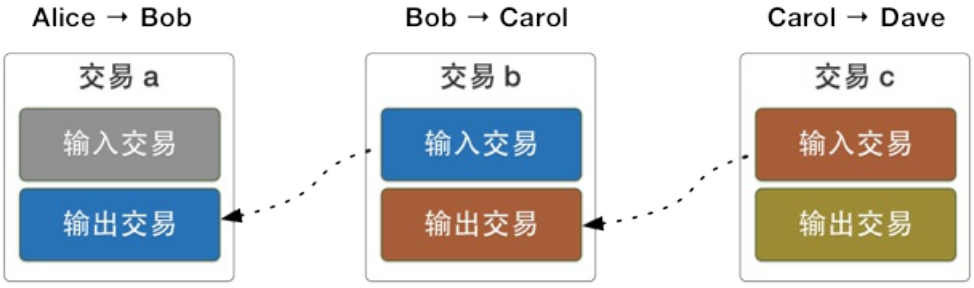
假设有这么一系列交易[3]：



1. 上图的三个交易都是单输入单输出交易
2. 每个『输入交易』『输出交易』中，都包含对应的『脚本』
   3.交易**a**，Alice转账给Bob；交易**b**，Bob转账给Carol；交易**c**，Carol转账给Dave
3. 当前交易的『输入』都引用前一个交易的『输出』，如交易b的『输入』引用交易a的『输出』

按照之前的说法，交易**a**中的『输出脚本』就是Alice为Bob出的数学题。那么，Bob想要引用交易**a**『输出交易』的比特币，就要解开这道数学题。题解是在交易**b**的『输入脚本』里给出的！Bob解开了这道题，获得了奖金，然后在交易**b**中为Carol出一道数学题，等待Carol来解…

所以说，下图中相同颜色的『输出』和『输入』才是一对题和解：



脚本语言

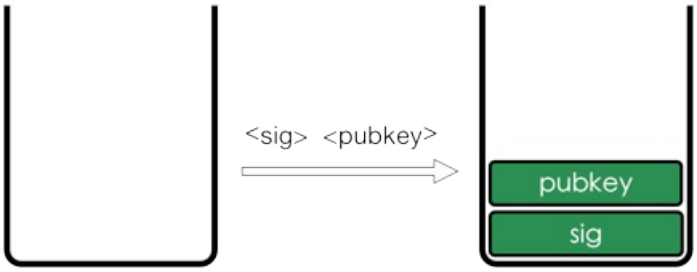
Bitcoin Wiki给出的对脚本的解释：

> 比特币在交易中使用脚本系统，与FORTH(一种编译语言)一样，脚本是简单的、基于堆栈的、并且从左向右处理，它特意设计成非图灵完整，没有LOOP语句。

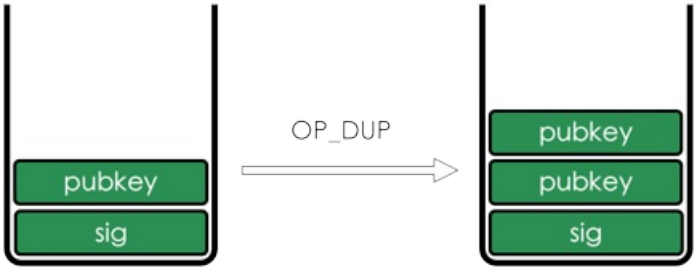
要理解比特币脚本，先要了解『堆栈』，这是一个后进先出(Last In First Out)的容器，脚本系统对数据的操作都是通过它完成的。比特币脚本系统中有两个堆栈：主堆栈和副堆栈，一般来说主要使用主堆栈。举几个简单的例子，看下指令是如何对堆栈操作的（完整的指令集在Wiki里可以找到）：

下面来看下这两段脚本是如何执行，来完成『解题』过程的。

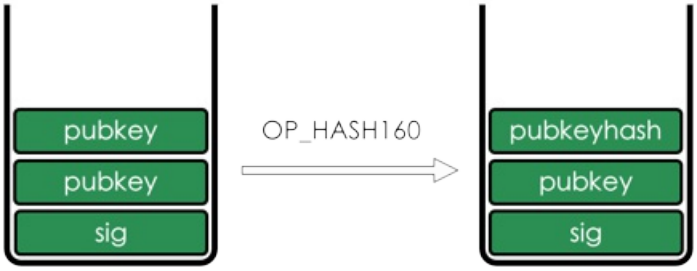
1. 首先执行的是『输入脚本』。因为脚本是从左向右执行的，那么先入栈的是『签名』此签名是交易b中Bob的签名，，随后是『公钥』，Bob的公钥（能通过HASH160得到交易a中输出交易的地址）



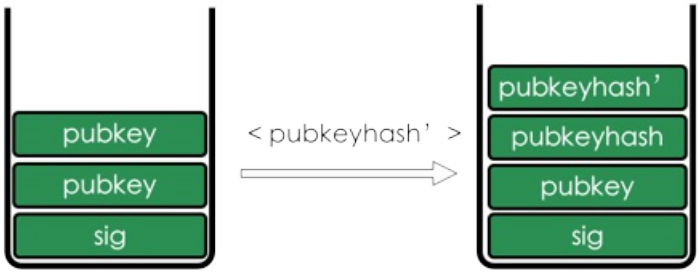
2.接着，执行的是『输出脚本』交易a的输出脚本。从左向右执行，第一个指令是OP_DUP——复制栈顶元素



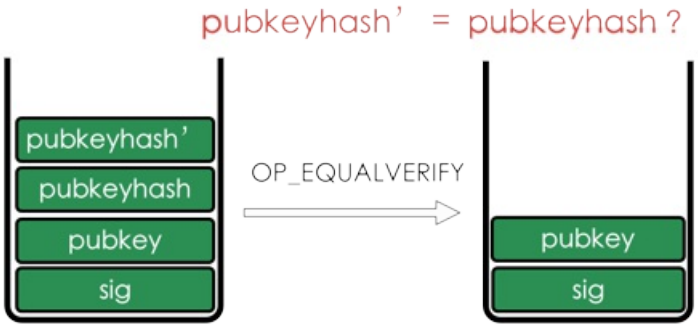
3.OP_HASH160——计算栈顶元素(Bob公钥的)Hash，得到pubkeyhash



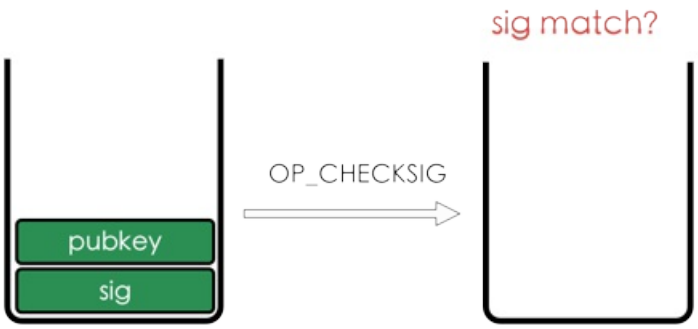
4.将交易a『输出脚本』中的『公钥哈希』入栈，为了和前面计算得到的哈希区别，称它为pubkeyhash'



5.OP_EQUALVERIFY——检查栈顶前两元素是否相等(判断地址交易a中转向的地址是否就是现在交易b中的输入脚本的地址)，如果相等继续执行，否则中断执行，返回失败



6.OP_CHECKSIG——使用栈顶前两元素执行签名校验操作(在地址正确的前提下，再校验Bob是否拥有该地址的私钥，也就是能否给出正确的签名)，如果相等，返回成功，否则返回失败



这样一串指令执行下来，就可以验证这道数学题是否做对了，也就是说验明了想要花费『钱包地址』中比特币的人是否拥有对应的『私钥』。上面的执行过程是可以在脚本模拟器中执行的，能够看到每一步执行的状态，感兴趣的童鞋可以尝试一下。

[1]. 转自 汪海波Hyper的理解比特币脚本 ↩

[2]. 转自 Pony小马的谈谈自己对比特币脚本的理解 ↩

[3]. Alice转给Bob比特币的数量是X，意味着Bob有X个UTXO\(Unspent Transaction Output\)，Bob可以用这UTXO转账给Carol,那为啥Bob就能用Alice转给他的那X个比特币呢？因为交易a的输出交易中，写明了Bob的地址以及转移的比特币数量。↩

第四节 交易脚本语言操作符，常量和符号

以下的表和描述参见https://en.bitcoin.it/wiki/Script

表1.脚本压入堆栈

符号	值 (十六进制)	描述
OP_0 or OP_FALSE	0x00	一个字节空串被压入堆栈中
1-75	0x01-0x4b	把接下来的N 个字节压入堆栈中，N 的取值在1 到75 之间
OP_PUSHDATA1	0x4c	下一个脚本字节包括N，会将接下来的N 个字节压入堆栈
OP_PUSHDATA2	0x4d	下两个脚本字节包括N，会将接下来的N 个字节压入堆栈
OP_PUSHDATA4	0x4e	下四个脚本字节包括N，会将接下来的N 个字节压入堆栈
OP_1NEGATE	0x4f	将脚本-1 压入堆栈
OP_RESERVED	0x50	终止- 交易无效（除非在未执行的OP_IF语句中）
OP_1 or OP_TRUE	0x51	将脚本1 压入堆栈
OP_2 to OP_16	0x52 to 0x60	将脚本N 压入堆栈，例如OP_2 压入脚本“2”

表2.有条件的流控制的操作符

符号	值 (十六进制)	描述
OP_NOP	0x61	无操作
OP_VER	0x62	终止- 交易无效（除非在未执行的OP_IF 语句中）
OP_IF	0x63	如果栈顶元素值为0，语句将被执行
OP_NOTIF	0x64	如果栈顶元素值不为0，语句将被执行
OP_VERIF	0x65	终止- 交易无效
OP_VERNOTIF	0x66	终止- 交易无效
OP_ELSE	0x67	如果前述的OP_IF 或OP_NOTIF 或OP_ELSE 未被执行，这些语句就会被执行
OP_ENDIF	0x68	终止OP_IF, OP_NOTIF, OP_ELSE 区块
OP_VERIFY	0x69	如果栈顶元素值非真，则标记交易无效
OP_RETURN	0x6a	标记交易无效

表3.时间锁操作符

符号	值 (十六进制)	描述
OP_CHECKLOCKTIMEVERIFY (previously OP_NOP2)	0xb1	如果栈顶元素比交易锁定时间字段大，则将交易标记为无效。否则脚本评测将像OP_NOP操作一样继续执行。交易在一下4种之一的情况下是无效的：1.堆栈是空的；2.栈顶元素是负数；3.当交易锁定时间字段值少于500000000时，栈顶元素大于等于500000000，反之亦然；4.输入序列字段等于0xffffffff。具体内容详见BIP-65。
OP_CHECKSEQUENCEVERIFY (previously OP_NOP3)	0xb2	如果输入值(BIP 0068强制规定的顺序)的相对锁定时间不等于或多于栈顶元素值时，将交易标记为无效。具体内容详见BIP-112。

表4.堆栈操作符

符号	值 (十六进制)	描述
OP_TOALTSTACK	0x6b	从主堆栈中取出元素，推入辅堆栈。
OP_FROMALTSTACK	0x6c	从辅堆栈中取出元素，推入主堆栈
OP_2DROP	0x6d	移除栈顶两个元素
OP_2DUP	0x6e	复制栈顶两个元素
OP_3DUP	0x6f	复制栈顶三个元素

OP_2OVER	0x70	把栈底的第三、第四个元素拷贝到栈顶
OP_2ROT	0x71	移动第五、第六元素到栈顶
OP_2SWAP	0x72	将栈顶的两个元素进行交换
OP_IFDUP	0x73	如果栈顶元素值不为0，复制该元素值
OP_DEPTH	0x74	Count the items on the stack and push the resulting count
OP_DROP	0x75	删除栈顶元素
OP_DUP	0x76	复制栈顶元素
OP_NIP	0x77	删除栈顶的下一个元素
OP_OVER	0x78	复制栈顶的下一个元素到栈顶
OP_PICK	0x79	把堆栈的第n 个元素拷贝到栈顶
OP_ROLL	0x7a	把堆栈的第n 个元素移动到栈顶
OP_ROT	0x7b	翻转栈顶的三个元素
OP_SWAP	0x7c	栈顶的三个元素交换
OP_TUCK	0x7d	拷贝栈顶元素并插入到栈顶第二个元素之后

表5.字符串接操作

符号	值 (十六进制)	描述
OP_CAT	0x7e	连接两个字符串，已禁用
OP_SUBSTR	0x7f	返回字符串的一部分，已禁用
OP_LEFT	0x80	在一个字符串中保留左边指定长度的子串，已禁用
OP_RIGHT	0x81	在一个字符串中保留右边指定长度的子串，已禁用
OP_SIZE	0x82	把栈顶元素的字符串长度压入堆栈

表6.二进制算术和条件

符号	值 (十六进制)	描述
OP_INVERT	0x83	所有输入的位取反，已禁用
OP_AND	0x84	对输入的所有位进行布尔与运算，已禁用
OP_OR	0x85	对输入的每一位进行布尔或运算，已禁用
OP_XOR	0x86	对输入的每一位进行布尔异或运算，已禁用
OP_EQUAL	0x87	如果输入的两个数相等，返回1，否则返回0
OP_EQUALVERIFY	0x88	与OP_EQUAL 一样，如结果为0，之后运行OP_VERIFY
OP_RESERVED1	0x89	终止- 无效交易（除非在未执行的OP_IF 语句中）
OP_RESERVED2	0x8a	终止-无效交易（除非在未执行的OP_IF 语句中）

表7.数值操作

符号	值 (十六进制)	描述
OP_1ADD	0x8b	栈顶值加1
OP_1SUB	0x8c	栈顶值减1
OP_2MUL	0x8d	无效（栈顶值乘2）
OP_2DIV	0x8e	无效（栈顶值除2）
OP_NEGATE	0x8f	栈顶值符号取反
OP_ABS	0x90	栈顶值符号取正
OP_NOT	0x91	如果栈顶值为0 或1，则输出1或0；否则输出0
OP_0NOTEQUAL	0x92	输入值为0 输出0；否则输出1
OP_ADD	0x93	弹出栈顶的两个元素，压入二者相加结果

OP_SUB	0x94	弹出栈顶的两个元素，压入二者相减（第二项减去第一项）结果
OP_MUL	0x95	禁用（栈顶两项的积）
OP_DIV	0x96	禁用（输出用第二项除以第一项的倍数）
OP_MOD	0x97	禁用（输出用第二项除以第一项得到的余数）
OP_LSHIFT	0x98	禁用（左移第二项，移动位数为第一项的二进制位数）
OP_RSHIFT	0x99	禁用（右移第二项，移动位数为第一项的二进制位数）
OP_BOOLAND	0x9a	布尔与运算，两项都不为0，输出1，否则输出0
OP_BOOLOR	0x9b	布尔或运算，两项有一个不为0，输出1，否则输出0
OP_NUMEQUAL	0x9c	两项相等则输出1，否则输出为0
OP_NUMEQUALVERIFY	0x9d	与NUMEQUAL 相同，如结果为0运行OP_VERIFY
OP_NUMNOTEQUAL	0x9e	如果栈顶两项不是相等数的话，则输出1
OP_LESSTHAN	0x9f	如果第二项小于栈顶项，则输出1
OP_GREATERTHAN	0xa0	如果第二项大于栈顶项，则输出1
OP_LESSTHANOREQUAL	0xa1	如果第二项小于或等于第一项，则输出1
OP_GREATERTHANOREQUAL	0xa2	如果第二项大于或等于第一项，则输出1
OP_MIN	0xa3	输出栈顶两项中较小的一项
OP_MAX	0xa4	输出栈顶两项中较大的一项
OP_WITHIN	0xa5	如果第三项的数值介于前两项之间，则输出1

表8.加密和散列操作

符号	值 (十六进制)	描述
OP_RIPEMD160	0xa6	返回栈顶项的RIPEMD160 哈希值
OP_SHA1	0xa7	返回栈顶项SHA1 哈希值
OP_SHA256	0xa8	返回栈顶项SHA256 哈希值
OP_HASH160	0xa9	栈顶项进行两次HASH，先用SHA-256，再用RIPEMD-160
OP_HASH256	0xaa	栈顶项用SHA-256 算法HASH 两次
OP_CODESEPARATOR	0xab	标记已进行签名验证的数据
OP_CHECKSIG	0xac	交易所用的签名必须是哈希值和公钥的有效签名，如果为真，则返回1
OP_CHECKSIGVERIFY	0xad	与CHECKSIG 一样，但之后运行OP_VERIFY
OP_CHECKMULTISIG	0xae	对于每对签名和公钥运行CHECKSIG。所有的签名要与公钥匹配。因为存在BUG，一个未使用的外部值会从堆栈中删除。
OP_CHECKMULTISIGVERIFY	0xaf	与CHECKMULTISIG 一样，但之后运行OP_VERIFY

表9.非操作符

符号	值 (十六进制)	描述
OP_NOP1-OP_NOP10	0xb0-0xb9	无操作忽略

表10.仅供内部使用的保留关键字

符号	值 (十六进制)	描述
OP_SMALLDATA	0xf9	代表小数据域
OP_SMALLINTEGER	0xfa	代表小整数数据域
OP_PUBKEYS	0xfb	代表公钥域
OP_PUBKEYHASH	0xfd	代表公钥哈希域
OP_PUBKEY	0xfe	代表公钥域
OP_INVALIDOPCODE	0xff	代表当前未指定的操作码

第五节 比特币改进建议(BIPs)

比特币改进提案是向比特币社区提供信息的设计文档，或用于描述比特币的新功能，流程或环境。

根据BIP-01也就是BIP目的和指南(BIP Purpose and Guidelines)的规定，有三种BIP：

标准类BIP

描述影响大多数或所有比特币实现的任何更改，例如网络协议的更改，区块或交易有效性规则的更改，或影响使用比特币的应用程序的互操作性的任何更改或附加。

信息类BIP

描述比特币设计问题，或向比特币社区提供一般准则或信息，但不提出新功能。信息类BIP不一定代表比特币社区的共识或建议，因此用户和实施者可以忽略信息类BIP或遵循他们的建议。

过程类BIP

描述一个比特币过程，或者提出一个过程的更改(或一个事件)。过程类BIP类似于标准类BIP，但适用于比特币协议本身以外的其他领域。他们可能会提出一个实现，但不是比特币的代码库;他们经常需要社区的共识;与信息类BIP不同，它们不仅仅是建议，用户通常也不能随意忽略它们。例如包括程序，指南，决策过程的变化以及对比特币开发中使用的工具或环境的更改。任何元BIP也被视为一个过程BIP。

BIP记录在GitHub上的版本化存储库中：https://github.com/bitcoin/bips。下表BIP的快照显示在2017年4月BIP的快照。了解有关现有BIP及其内容的最新信息请咨询权威机构。

BIP# Title Owner Type Status
BIP-1 BIP Purpose and Guidelines Amir Taaki Process Replaced
BIP-2 BIP process, revised Luke Dashjr Process Active
BIP-8 Version bits with guaranteed lock-in Shaolin Fry Informational Draft
BIP-9 Version bits with timeout and delay Pieter Wuille, Peter Todd, Greg Maxwell, Rusty Russell Informational Final
BIP-10 Multi-Sig Transaction Distribution Alan Reiner Informational Withdrawn
BIP-11 M-of-N Standard Transactions Gavin Andresen Standard Final
BIP-12 OP_EVAL Gavin Andresen Standard Withdrawn
BIP-13 Address Format for pay-to-script-hash Gavin Andresen Standard Final
BIP-14 Protocol Version and User Agent Amir Taaki, Patrick Strateman Standard Final
BIP-15 Aliases Amir Taaki Standard Deferred
BIP-16 Pay to Script Hash Gavin Andresen Standard Final
BIP-17 OP_CHECKHASHVERIFY (CHV) Luke Dashjr Standard Withdrawn
BIP-18 hashScriptCheck Luke Dashjr Standard Proposed
BIP-19 M-of-N Standard Transactions (Low SigOp) Luke Dashjr Standard Draft
BIP-20 URI Scheme Luke Dashjr Standard Replaced
BIP-21 URI Scheme Nils Schneider, Matt Corallo Standard Final
BIP-22 getblocktemplate - Fundamentals Luke Dashjr Standard Final
BIP-23 getblocktemplate - Pooled Mining Luke Dashjr Standard Final
BIP-30 Duplicate transactions Pieter Wuille Standard Final
BIP-31 Pong message Mike Hearn Standard Final
BIP-32 Hierarchical Deterministic Wallets Pieter Wuille Informational Final
BIP-33 Stratized Nodes Amir Taaki Standard Draft
BIP-34 Block v2, Height in Coinbase Gavin Andresen Standard Final
BIP-35 mempool message Jeff Garzik Standard Final
BIP-36 Custom Services Stefan Thomas Standard Draft
BIP-37 Connection Bloom filtering Mike Hearn, Matt Corallo Standard Final
BIP-38 Passphrase-protected private key Mike Caldwell, Aaron Voisine Standard Draft
BIP-39 Mnemonic code for generating deterministic keys Marek Palatinus, Pavol Rusnak, Aaron Voisine, Sean Bowe Standard Proposed
BIP-40 Stratum wire protocol Marek Palatinus Standard BIP number allocated BIP-41 Stratum mining protocol Marek Palatinus Standard BIP number allocated BIP-42 A finite monetary supply for Bitcoin Pieter Wuille Standard Draft
BIP-43 Purpose Field for Deterministic Wallets Marek Palatinus, Pavol Rusnak Informational Draft
BIP-44 Multi-Account Hierarchy for Deterministic Wallets Marek Palatinus, Pavol Rusnak Standard Proposed
BIP-45 Structure for Deterministic P2SH Multisignature Wallets Manuel Araoz, Ryan X. Charles, Matias Alejo Garcia Standard Proposed
BIP-47 Reusable Payment Codes for Hierarchical Deterministic Wallets Justus Ranvier Informational Draft
BIP-49 Derivation scheme for P2WPKH-nested-in-P2SH based accounts Daniel Weigl Informational Draft
BIP-50 March 2013 Chain Fork Post-Mortem Gavin Andresen Informational Final
BIP-60 Fixed Length "version" Message (Relay-Transactions Field) Amir Taaki Standard Draft
BIP-61 Reject P2P message Gavin Andresen Standard Final
BIP-62 Dealing with malleability Pieter Wuille Standard Withdrawn
BIP-63 Stealth Addresses Peter Todd Standard BIP number allocated BIP-64 getutxo message Mike Hearn Standard Draft

BIP-65 OP_CHECKLOCKTIMEVERIFY Peter Todd Standard Final
BIP-66 Strict DER signatures Pieter Wuille Standard Final
BIP-67 Deterministic Pay-to-script-hash multi-signature addresses through public key sorting Thomas Kerin, Jean-Pierre Rupp, Ruben de Vries Standard Proposed
BIP-68 Relative lock-time using consensus-enforced sequence numbers Mark Friedenbach, BtcDrak, Nicolas Dorier, kinoshitajona Standard Final
BIP-69 Lexicographical Indexing of Transaction Inputs and Outputs Kristov Atlas Informational Proposed
BIP-70 Payment Protocol Gavin Andresen, Mike Hearn Standard Final
BIP-71 Payment Protocol MIME types Gavin Andresen Standard Final
BIP-72 bitcoin: uri extensions for Payment Protocol Gavin Andresen Standard Final
BIP-73 Use "Accept" header for response type negotiation with Payment Request URLs Stephen Pair Standard Final
BIP-74 Allow zero value OP_RETURN in Payment Protocol Toby Padilla Standard Draft
BIP-75 Out of Band Address Exchange using Payment Protocol Encryption Justin Newton, Matt David, Aaron Voisine, James MacWhyte Standard Draft
BIP-80 Hierarchy for Non-Colored Voting Pool Deterministic Multisig Wallets Justus Ranvier, Jimmy Song Informational Deferred
BIP-81 Hierarchy for Colored Voting Pool Deterministic Multisig Wallets Justus Ranvier, Jimmy Song Informational Deferred
BIP-83 Dynamic Hierarchical Deterministic Key Trees Eric Lombrozo Standard Draft
BIP-90 Buried Deployments Suhas Daftuar Informational Draft
BIP-99 Motivation and deployment of consensus rule changes ([soft/hard]forks) Jorge Timón Informational Draft
BIP-101 Increase maximum block size Gavin Andresen Standard Withdrawn
BIP-102 Block size increase to 2MB Jeff Garzik Standard Draft
BIP-103 Block size following technological growth Pieter Wuille Standard Draft
BIP-104 'Block75' - Max block size like difficulty t.khan Standard Draft
BIP-105 Consensus based block size retargeting algorithm BtcDrak Standard Draft
BIP-106 Dynamically Controlled Bitcoin Block Size Max Cap Upal Chakraborty Standard Draft
BIP-107 Dynamic limit on the block size Washington Y. Sanchez Standard Draft
BIP-109 Two million byte size limit with sigop and sighash limits Gavin Andresen Standard Rejected
BIP-111 NODE_BLOOM service bit Matt Corallo, Peter Todd Standard Proposed
BIP-112 CHECKSEQUENCEVERIFY BtcDrak, Mark Friedenbach, Eric Lombrozo Standard Final
BIP-113 Median time-past as endpoint for lock-time calculations Thomas Kerin, Mark Friedenbach Standard Final
BIP-114 Merkelized Abstract Syntax Tree Johnson Lau Standard Draft
BIP-120 Proof of Payment Kalle Rosenbaum Standard Draft
BIP-121 Proof of Payment URI scheme Kalle Rosenbaum Standard Draft
BIP-122 URI scheme for Blockchain references / exploration Marco Pontello Standard Draft
BIP-123 BIP Classification Eric Lombrozo Process Active
BIP-124 Hierarchical Deterministic Script Templates Eric Lombrozo, William Swanson Informational Draft
BIP-125 Opt-in Full Replace-by-Fee Signaling David A. Harding, Peter Todd Standard Proposed
BIP-126 Best Practices for Heterogeneous Input Script Transactions Kristov Atlas Informational Draft
BIP-130 sendheaders message Suhas Daftuar Standard Proposed
BIP-131 "Coalescing Transaction" Specification (wildcard inputs) Chris Priest Standard Draft
BIP-132 Committee-based BIP Acceptance Process Andy Chase Process Withdrawn
BIP-133 feefilter message Alex Morcos Standard Draft
BIP-134 Flexible Transactions Tom Zander Standard Draft
BIP-140 Normalized TXID Christian Decker Standard Draft
BIP-141 Segregated Witness (Consensus layer) Eric Lombrozo, Johnson Lau, Pieter Wuille Standard Draft
BIP-142 Address Format for Segregated Witness Johnson Lau Standard Deferred
BIP-143 Transaction Signature Verification for Version 0 Witness Program Johnson Lau, Pieter Wuille Standard Draft
BIP-144 Segregated Witness (Peer Services) Eric Lombrozo, Pieter Wuille Standard Draft
BIP-145 getblocktemplate Updates for Segregated Witness Luke Dashjr Standard Draft
BIP-146 Dealing with signature encoding malleability Johnson Lau, Pieter Wuille Standard Draft
BIP-147 Dealing with dummy stack element malleability Johnson Lau Standard Draft
BIP-148 Mandatory activation of segwit deployment Shaolin Fry Standard Draft
BIP-150 Peer Authentication Jonas Schnelli Standard Draft
BIP-151 Peer-to-Peer Communication Encryption Jonas Schnelli Standard Draft
BIP-152 Compact Block Relay Matt Corallo Standard Draft
BIP-171 Currency/exchange rate information API Luke Dashjr Standard Draft
BIP-180 Block size/weight fraud proof Luke Dashjr Standard Draft
BIP-199 Hashed Time-Locked Contract transactions Sean Bowe, Daira Hopwood Standard Draft

闪电网络

LND Overview and Developer Guide https://github.com/bcongdon/awesome-lightning-network/blob/master/readme.md
https://dev.lightning.community/tutorial/01-lncli/index.html

比特币基本概念

Coursera Bitcoin and Cryptocurrency Technologies Bitcoin Developer Guide **locktime**：One thing all signature hash types sign is the transaction's locktime. (Called nLockTime in the Bitcoin Core source code.) The locktime indicates the earliest time a transaction can be added to the block chain. 通常被设置为0，表示transaction一创建好就马上发送到比特币网络，带locktime的transaction表示交易只有在未来locktime的某个时刻才被验证并添加到比特币网络上去。这个时间的设置保证交易者有足够的改变主意的时间。If less than 500 million, locktime is parsed as a block height. The transaction can be added to any block which has this height or higher. If greater than or equal to 500 million, locktime is parsed using the Unix epoch time format (the number of seconds elapsed since 1970-01-01T00:00 UTC—currently over 1.395 billion). The transaction can be added to any block whose block time is greater than the locktime.

Hash Time-Locked Contracts (HTLCs)

Neo智能合约开发

Neo Smart Contract](http://docs.neo.org/en-us/sc/quickstart/getting-started-csharp.html

1. 安装NeoContractPlugin
2. down load neo-compiler, compiler and publish neon
    i. set the neon.exe path to the system path
3. Create NeoContract project
4. Compile the Project

http://ndapp.org/ 上的基于Neo的DAPP一般就几十行代码，都是灌水的，没有啥学习的价值。还是靠Bitcoin的可以学到更多的技能。

DPos

石墨烯技术，也就是基于DPos的技术，

目前区块链生态系统主要分为三类，一类是比特币生态系，一类是以太坊生态系，而另外一种就是石墨烯生态系。

比特币生态包括 BTC 以及其数量众多的分叉币，BTC 是加密数字货币的开山鼻祖，拥有最为广泛的共识。以太坊生态系又叫做 ERC20 Token，CoinMarketCap 上绝大多数的 Token 都是基于以太坊 ERC20。以太坊生态提供的智能合约，可以极为简便的发行 Token，项目再利用 ICO 的方式进行快速融资。石墨烯生态的代表有 BTS，Steem 和 EOS。石墨烯采用的是 DPOS 的共识机制，出快速度大约为 1.5s，石墨烯技术使得区块链应用更高的交易吞吐量，BTS 可以处理十万级别的 TPS，而 EOS 则是宣称百万级别的 TPS。同时石墨烯技术高并发处理能力也是比特币和 ETH 无法做到的。

石墨烯是区块链工具组，由**Cryptonomex**公司开发，GitHub 项目地址：https://github.com/cryptonomex/graphene

，采用 C++ 编写，丹尼尔 • 拉里默（Dan Larimer）是 Cryptonomex 的创始人，而他的父亲斯坦•拉里默（Stan Larimer）是 Cryptonomex 的主席。Cryptonomex 基本上都是在石墨烯区块链库基础上做开发的，石墨烯区块链库已经被多个区块链所采纳，比如 BitShares，Muse，Identabit，Play 等。[1] 与大多数数字货币类似，Graphene (石墨烯) 使用区块链来记录参与者的转账信息及市场行为。由于每个区块总是指向前一个区块，因此一个区块链条包含了所有在网络上发生的交易信息。区块链是一个公开的、可审计的账簿，每个人都能够查看详细数据，并验证交易、市场订单和买卖盘数据。

比特币钱包开发

基于C#的比特币钱包
https://github.com/nopara73/DotNetWallet/blob/master/README.md

HBitcoin.KeyManagement是什么？

初识NuGet - 概念，安装和使用

NuGet中使用的dll位于：C:\Users\pili.nuget

BitcoinLib The most complete, up-to-date, battle-tested Library and RPC Wrapper for Bitcoin, Litecoin, Dogecoin and Bitcoin-Clones in C#.

HBitcoin 基于NBitcoin,因此最终开发还是基于NBitcoin，我们得写一些调用NBitcoin接口的文档来辅助我们开发，先用C#最速快发一款windows下使用的UI钱包，再移植到QT上面（得想想在QT上还能调用NBitcoin吗？这是个问题，不急，先熟悉了钱包开发的流程再说，从简单的功能入手，一步步构建完美的产品）。(Linux是支持dotnet core的)

一切不是那么如意，但是要坚持下去，去看别人看不见的风景。

NBitcoin参考文档

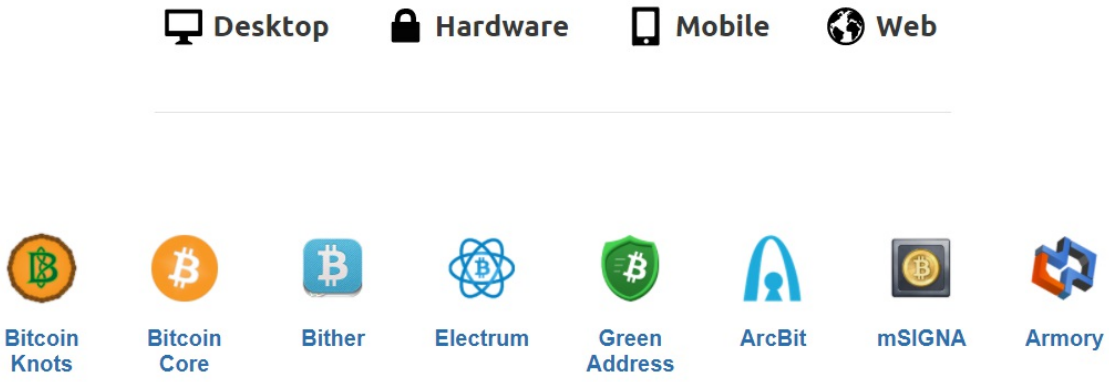
https://www.codeproject.com/Articles/835098/NBitcoin-Build-Them-All#simple

https://www.codeproject.com/Articles/768412/NBitcoin-The-most-complete-Bitcoin-port-Part-Crypt

https://github.com/ProgrammingBlockchain/ProgrammingBlockchainCodeExamples

NBitcoin: How to make your first transaction with NBitcoin https://www.youtube.com/watch?v=X4ZwRWIF49w

比特币钱包比较



https://www.zhihu.com/question/21478404

http://bitkan.com/news/topic/6114?bkfrom=appshare&bktarget=weibo&bkuserid=6923

哪个是最佳的比特币钱包？

基于N...开发

创建私钥，得到公钥

```
RandomUtils.Random = new UnsecureRandom();
Key privateKey = new Key();//私钥
PubKey publicKey = privateKey.PubKey;
txb_public_key.Text = publicKey.GetAddress(Network.TestNet).ToString();
txb_private_key.Text = privateKey.ToString(Network.TestNet);//输出
```

A Bitcoin wallet must do the following:

1. Generate addresses.
2. Recognize transactions spent to these addresses.
3. Detect transactions, those are spending from these addresses.
4. Show the history of the transactions involving this wallet.
5. Handle reorgs.
6. Handle conflicts.
7. Dynamically calculate transaction fees.
8. Build and sign transactions.
9. Broadcast transactions.

温国兵的随想录
区块链钱包开发
本文讲解了开发钱包的预备知识，包括第一是什么是钱包，以及相关的分类，第二是 RPC、JSON-RPC 以及 JSON，第三是了解区块链相关的基础知识，第四是掌握一门开发语言。接着浅谈了怎么样开发，最后列出了主流项目相关的 RPC 接口以及开源钱包项目。如果读者对钱包开发感兴趣，希望本文能够给读者一个指引。

20180527

通过密码设定来获取助记词：

Safe safe = Safe.Create(out mnemonic, pw, walletFilePath, Config.Network);

MCCBit

Password:AA

mnemonic:people relief fade phone unveil bracket change say enter term add worth

DotNetWallet:

password:Alpha2018

mnemonic: scan jazz seminar dog joke caution print hurdle purse merge grape impose

Sent!

TX ID: bb0de9e20df766655ef10466c7b9e761517ce20a70985ee8c2e818bdf5d58ac8

testnet donate: **2N8hwP1WmJrFF5QWABn38y63uYLhnJYJYTF**

password: Alpha2018Simon

NBitcoin.RPC

我们能做的所有查询都是基于Bitcoin Core's RPC API

遇到的问题：

{"Could not load file or assembly 'StandardConfiguration, Version=1.0.0.14, Culture=neutral, PublicKeyToken=null' or one of its dependencies. The system cannot find the file specified.":"StandardConfiguration, Version=1.0.0.14, Culture=neutral, PublicKeyToken=null"}

Solution：usingNuGet Package Manager: install NicolasDorier.StandardConfiguration.1.0.0.14

"127.0.0.1:24445"

"Unable to connect to the remote server"

{"No connection could be made because the target machine actively refused it 127.0.0.1:24445"}

{"Response status code does not indicate success: 401 (Unauthorized)."}

"http://127.0.0.1:24445/v1/cryptos/BTC/derivations/tpubD6NzVbkrYhZ4Wj8xtqtoehXE88zBZbLkAhRuScECu8ukFxD6Y8T5VHcECy5noySj9oVY8baBRcKUjh5xq9sCudrxAvCY87jomtdkbUavHUV-\[legacy\]/utxos?longPolling=False"

基于Nbitcoin的比特币钱包开发日志

- [ ] Test code以及Example code和NBitcoin code一起看，因为最终调用的地方还是在NBitcoin

20180524

- [ ] 要开发出一套基于regtest网络的钱包，当然是用于测试。
- [ ] 而后是testnet与main net.
- [ ] 每天记录开发进度

20180526

- [ ] 钱包开发就是基于虚拟币自身的RPC API进行开发。
  - [ ] 可以调用别人的库，比如.Net上的NBitcoin
  - [ ] 也可以直接调用源码，这两种方式都该考虑如何开发。
  - [ ] 在Windows上安装好了Qt,下一步开发钱包

20180527

- [ ] 钱包开发，基于NBitcoin

第五章:算法

第一节 Kriging插值

https://xg1990.com/blog/archives/222

空间插值问题，就是在已知空间上若干离散点 (x_i, y_i)的某一属性(如气温，海拔)的观测值$z_i = z(x_i, y_i)$的条件下，估计空间上任意一点(x, y)的属性值的问题。
直观来讲，根据地理学第一定律，

大意就是，地理属性有空间相关性，相近的事物会更相似。由此人们发明了反距离插值，对于空间上任意一点(x, y) 的属性$z = z(x, y)$ ，

定义反距离插值公式估计量 $\hat{z} = \sum_{i=0}^{n} \frac{1}{d^{\alpha}} z_i$
其中α通常取1或者2。

即，用空间上所有已知点的数据加权求和来估计未知点的值，权重取决于距离的倒数(或者倒数的平方)。

那么，距离近的点，权重就大;距离远的点，权重就小。反距离插值可以有效的基于地理学第一定律估计属性值空间分布，但仍然存在很多问题:
α的值不确定 用倒数函数来描述空间关联程度不够准确

因此更加准确的克里金插值方法被提出来了

克里金插值公式$\hat{z_o} = \sum_{i=0}^{n} \lambda_i z_i$

其中$\hat{z_0}$是点(x_o, y_o)处的估计值，即:$z_o = z(x_o, y_o)$
假设条件:

1. 无偏约束条件 $E(\hat{z_0} - z_0) = 0$
2. 优化目标/代价函数 $J = Var((\hat{z_0} - z_0))$取极小值
3. 半方差函数r_{ij}与空间距离d_{ij}存在关联,并且这个关联可以通过这这两组数拟合出来，因此可以用距离d_{ij}来求得r_{ij}

半方差函数$r_{ij} = \sigma^2 - C_{ij}$; 等价于$r_{ij} = \frac{1}{2} E[(z_i - z_j)^2]$

其中:$C_{ij} = Cov(z_i, z_j) = Cov(R_i, R_j)$

求得r_{ij}之后，我们就可以求得λ_i

$$\begin{bmatrix} r_{11} & r_{12} & \cdots & r_{1n} & 1 \\ r_{21} & r_{22} & \cdots & r_{2n} & 1 \\ \cdots & \cdots & \cdots & \cdots & \cdots \\ r_{n1} & r_{n2} & \cdots & r_{nn} & 1 \\ 1 & 1 & \cdots & 1 & 0 \end{bmatrix} \begin{bmatrix} \lambda_1 \\ \lambda_2 \\ \cdots \\ \lambda_n \\ 0 \end{bmatrix} = \begin{bmatrix} r_{1o} \\ r_{2o} \\ \cdots \\ r_{no} \\ 1 \end{bmatrix}$$

最长递增子序列

对于序列5, 3, 4, 8, 6, 7, 其最长的递增子序列是3, 4, 6, 7,这里, 不需要保证元素是连在一起的, 只需要序号上升即可。问题变成求解n元系列a[1], a[2], a[3],...,a[n],

对于i < j < n, 假设我们已经求得前i个元素的最长递增子序列长度为MaxLen[i] 满足i < j, 那么我们可以遍历i从1到j-1来求得MaxLen[j]。基本的一些小算法写在一个CPP里面, 最后封装成一个类,一般要设计成模板类。

```
int Algo::LongestIncreaseSubsequence(int arr[], int arrSize)
{
    int *maxLen = new int[arrSize]();
    for (int i =0; i<arrSize; i++)
    {
        maxLen[i] = 1;
    }
    for (int j = 1; j < arrSize; j++)
    {
        int maxLenJ = 1;
        for (int i=0; i<j; i++)
        {
            if (arr[i] < arr[j])
            {
                maxLenJ = maxLen[i] + 1;
                maxLen[j] = std::max({ maxLen[j] ,maxLenJ });
            }
        }
    }
    return maxLen[arrSize-1];
}
```

LU分解

任何非奇异方阵都可以分解成上三角阵与下三角阵之积

$$\begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \cdots & \cdots & \cdots & \cdots \\ a_{n1} & a_{n2} & \cdots & a_{nn} \end{bmatrix} = \begin{bmatrix} l_{11} & 0 & \cdots & 0 \\ l_{21} & l_{22} & \cdots & 0 \\ \cdots & \cdots & \cdots & \cdots \\ l_{n1} & l_{n2} & \cdots & l_{nn} \end{bmatrix} * \begin{bmatrix} u_{11} & u_{12} & \cdots & u_{1n} \\ 0 & u_{22} & \cdots & a_{2n} \\ \cdots & \cdots & \cdots & \cdots \\ 0 & 0 & \cdots & u_{nn} \end{bmatrix}$$

对比两边矩阵的，可以求得：

但是编程时，行列都是从0开始时，要注意转换。

第1行：$a_{1j} = u_{1j}, j = 1, 2, .., n. => u_{1j} = a_{1j}$。

第1列：$a_{j1} = l_{j1}u_{11}, j = 1, 2, .., n. => l_{j1} = a_{j1}/u_{11}$。

...。

第k行：$a_{kj} = \sum_{i=1}^{i=k} l_{ki}u_{ij}, => u_{kj} = a_{kj} - \sum_{i=1}^{i=k-1} l_{ki}u_{ij}$。

第k列：$a_{jk} = \sum_{i=1}^{i=k} l_{ji}u_{ik}, => u_{jk} = [a_{jk} - \sum_{i=1}^{i=k-1} l_{ji}u_{ik}]/u_{kk}$。

因为前k-1行的u_{ij}都已知，前k-1列的l_{ij}都已知,因此可以求得第k行u_{ij}，第k列的l_{ij}。

问题：得保证a_{11}非0，以及矩阵非奇异。

利用LU分解求线性方程组的解

求解线性方程组 Ax=b相当于求解LUx=b;

设Y = UX; 因此LY = b;首先求解LY = b,

$$\begin{bmatrix} l_{11} & 0 & \cdots & 0 \\ l_{21} & l_{22} & \cdots & 0 \\ \cdots & \cdots & \cdots & \cdots \\ l_{n1} & l_{n2} & \cdots & l_{nn} \end{bmatrix} \begin{bmatrix} y_1 \\ y_2 \\ \cdots \\ y_n \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \\ \cdots \\ b_n \end{bmatrix}$$

求解上面的方程：

第1行：$y_1 = b_1$。

对于第k行：$b_k = \sum_{i=1}^{i=k} l_{ki}y_i => y_k = b_k - \sum_{i=1}^{i=k-1} l_{ki}y_i$。

求得Y之后，代入Y=UX求得X:

$$\begin{bmatrix} u_{11} & u_{12} & \cdots & u_{1n} \\ 0 & u_{22} & \cdots & u_{2n} \\ \cdots & \cdots & \cdots & \cdots \\ 0 & 0 & \cdots & u_{nn} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \cdots \\ x_n \end{bmatrix} = \begin{bmatrix} y_1 \\ y_2 \\ \cdots \\ y_n \end{bmatrix}$$

对于上三角矩阵，我们从第n行开始求解

对第n行：$y_n = u_{nn}x_n => x_n = y_n/u_{nn}$。

...。

对第k行：$y_k = \sum_{i=k}^{i=n} u_{ki}x_i => x_k = [y_n - \sum_{i=k+1}^{i=n} u_{ki}x_i]/u_{kk}$

这样通过LU分解矩阵就求得了线性方程组的就X.

矩阵求逆

我们可以利用LU分解来求非奇异方阵的逆矩阵。

AB=I

$$\begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \cdots & \cdots & \cdots & \cdots \\ a_{n1} & a_{n2} & \cdots & a_{nn} \end{bmatrix} * \begin{bmatrix} b_{11} & b_{12} & \cdots & b_{1n} \\ b_{21} & b_{22} & \cdots & b_{2n} \\ \cdots & \cdots & \cdots & \cdots \\ b_{n1} & b_{n2} & \cdots & b_{nn} \end{bmatrix} = \begin{bmatrix} 1 & 0 & \cdots & 0 \\ 0 & 1 & \cdots & 1 \\ \cdots & \cdots & \cdots & \cdots \\ 0 & 0 & \cdots & u_{nn} \end{bmatrix}$$

可以分解成求:

$$\begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \cdots & \cdots & \cdots & \cdots \\ a_{n1} & a_{n2} & \cdots & a_{nn} \end{bmatrix} * \begin{bmatrix} b_{1k} \\ b_{2k} \\ \cdots \\ b_{nk} \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ \cdots \\ 1 \\ \cdots \\ 0 \end{bmatrix}$$

右边的列，就只是第k行值非0；

$A * b_k = e_k$,对所有的e_k求出b_k就可以得到A的逆矩阵$A^{-1} = B$

实际求解中把A 换成LU来减少计算量。总的计算开销还是n^3。但是这样并不比直接的高斯消元法来的快。

算法实现如下。

```
//LU
template<class T>
void CMatrix<T>::LU(CMatrix &mat, int N, CMatrix* L, CMatrix* U)
{
    for (int k = 0; k <N; k++)
    {
        for (int j = k; j<N;j++)
        {
            T U_k_j = mat.Get(k, j);
            T L_j_k = mat.Get(j, k);
            for (int i = 0; i <k; i++)
            {
                U_k_j -= L->Get(k, i)*U->Get(i, j);
            }
            U->Set(k, j, U_k_j);

            for (int i = 0; i < k; i++)
            {
                L_j_k -= L->Get(j, i)*U->Get(i, k);
            }
            L_j_k = L_j_k / U->Get(k, k);
            L->Set(j, k, L_j_k);

        }
    }
}

//solve linear algebra equations
CVector Solve(CMatrix<double>& mat, CVector& vec)
{
    CVector X(vec.Size());
    CVector Y(vec.Size());
    if (mat.Columns() != mat.Rows() && mat.Rows() != vec.Size())
    {
        printf("Dimension not match!");
        return X;
    }
    CMatrix<double> L(vec.Size(), vec.Size());
    CMatrix<double> U(vec.Size(), vec.Size());
    mat.LU(mat, vec.Size(), &L, &U);
    Y = SolveLow(L, vec);
    cout << "Y\n" << Y;
    X = SolveUpper(U, Y);
    cout << "X\n" << X;
    return X;
}

CVector SolveLow(CMatrix<double>& mat, CVector& vec)
{
    CVector vecRes(vec.Size());
    if (mat.Columns() != mat.Rows() && mat.Rows() != vec.Size())
    {
        printf("Dimension not match!");
        return vecRes;
    }

    for (int k = 0; k < vec.Size(); k++)
    {
        double mRes = 0;
        mRes = vec.Get(k);
        for (int i = 0; i < k; i++)
        {
            mRes -= mat.Get(k, i)*vecRes.Get(i);
        }
        vecRes.Set(k, mRes);
    }
    return vecRes;
}

CVector SolveUpper(CMatrix<double>& mat, CVector& vec)
{
    CVector vecRes(vec.Size());
    if (mat.Columns() != mat.Rows() && mat.Rows() != vec.Size())
    {
        printf("Dimension not match!");
        return vecRes;
    }
```

```
    for (int k = vec.Size() -1; k >=0 ; k--)
    {
        double mRes = 0;
        mRes = vec.Get(k);
        for (int i = k+1; i <vec.Size(); i++)
        {
            mRes -= mat.Get(k, i)*vecRes.Get(i);
        }
        mRes = mRes / mat.Get(k, k);
        vecRes.Set(k, mRes);
    }
    return vecRes;
}

//inverse matrix
template<class T>
CMatrix<T>  CMatrix<T>::Inv()
{
    CMatrix result = *this;
    CMatrix<double> L(mRows, mColumns);
    CMatrix<double> U(mRows, mColumns);
    CMatrix<double> InvMat(mRows, mColumns);
    LU(result,mRows, &L, &U);
    for (int col = 0; col < mColumns; col++)
    {
        CVector vec(mRows, col);
        CVector mSolution(mRows);
        mSolution = SolveLow(L, vec);
        mSolution = SolveUpper(U, mSolution);
        for (int row = 0; row<mRows; row++)
        {
            InvMat.Set(row, col, mSolution.Get(row));
        }
    }
    return InvMat;
}
```

SVD

利用LU分解，我们可以求满秩的线性方程组的解。对于mxn（m>n）阶矩阵，对于超定方程（方程数目大于未知数的个数），因为一般没有解满足$Ax = b$,这就是最小二乘法发挥作用的地方。

这个问题的解就是使得：$||Ax - b||_2^2$值极小的解，通过求导（把x->x+e，求梯度等于0的x),可以得到解为：$x = (X^T A)^{-1} A^T b$

QR分解

定理：设A是mxn阶矩阵，m>=n,假设A为满秩的，则存在一个唯一的正交矩阵$Q(Q^T Q = I)$和唯一的具有正对角元$r_{ij} > 0$的nxn阶上三角阵R使得$A = QR$.

Gram-Schmidt正交化

Gram-Schmidt正交化的基本想法，是利用投影原理在已有基的基础上构造一个新的正交基。

$((\beta))_1$

http://elsenaju.eu/Calculator/QR-decomposition.htm

https://rosettacode.org/wiki/QR_decomposition

https://www.wikiwand.com/en/QR_decomposition#/Example_2

https://www.wikiwand.com/en/Householder_transformation

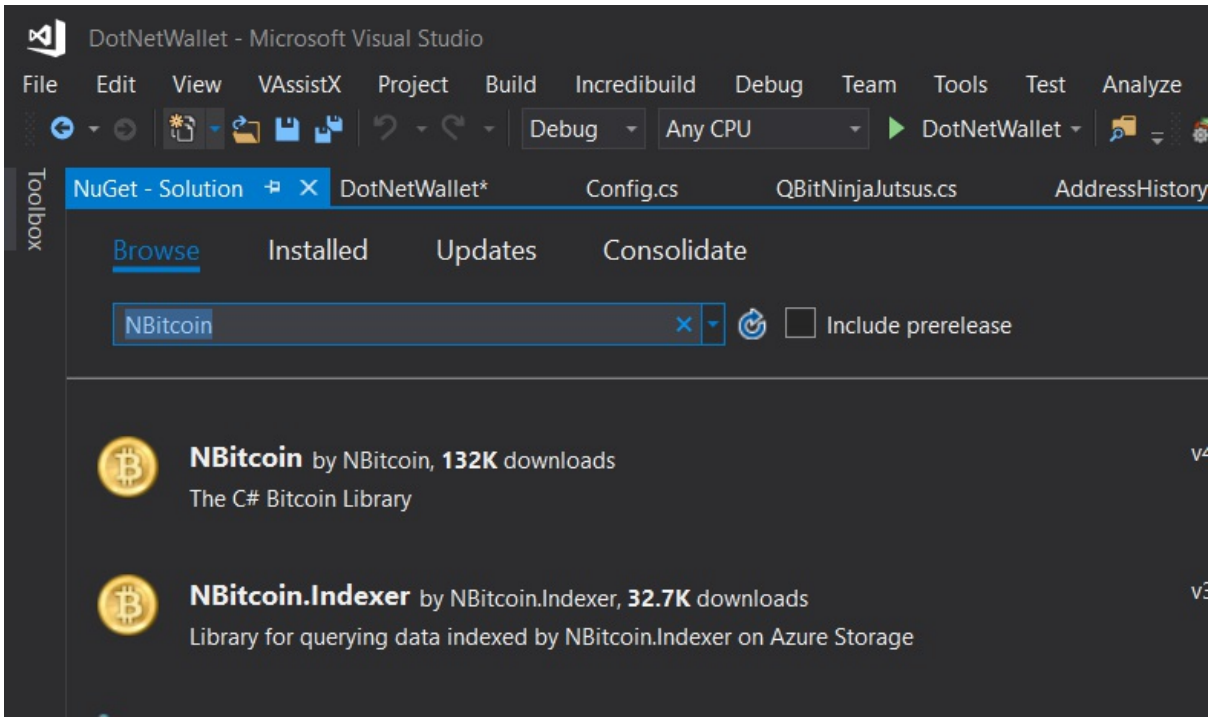
C#有开源的免费的代数库mathnet.numerics,功能也比较多，推荐通过Nuget来安装这个库

Nuget 安装dll

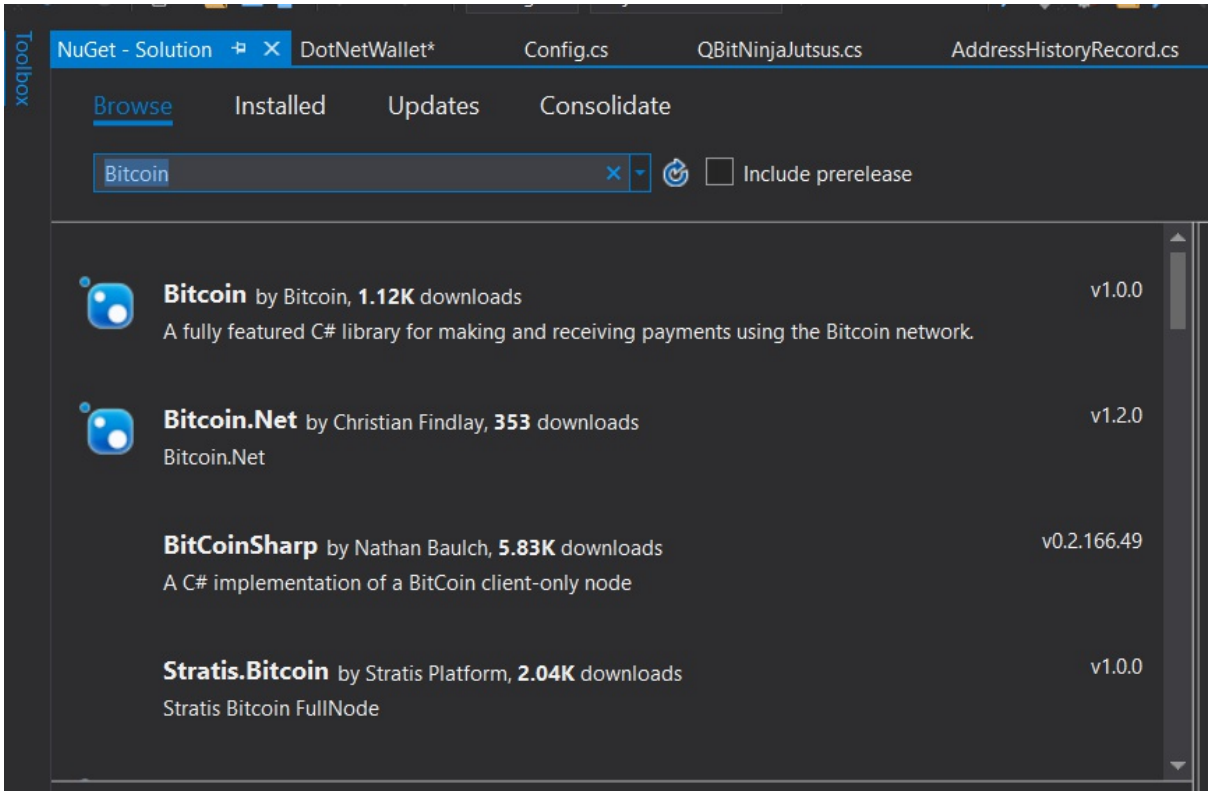
Tools-->Nuget Package Manager-->Manager Nuget packages for solution..-->Browse

https://numerics.mathdotnet.com/api/MathNet.Numerics.LinearAlgebra/Matrix`1.htm#QR

https://numerics.mathdotnet.com/matrix.html



研究这个


Toolbox
NuGet - Solution
DotNetWallet*
Config.cs
QBitNinjaJutsus.cs
AddressHistoryRecord.cs
Browse
Installed
Updates
Consolidate
Bitcoin
Include prerelease
Bitcoin by Bitcoin, 1.12K downloads
v1.0.0
A fully featured C# library for making and receiving payments using the Bitcoin network.
Bitcoin.Net by Christian Findlay, 353 downloads
v1.2.0
Bitcoin.Net
BitCoinSharp by Nathan Baulch, 5.83K downloads
v0.2.166.49
A C# implementation of a BitCoin client-only node
Stratis.Bitcoin by Stratis Platform, 2.04K downloads
v1.0.0
Stratis Bitcoin FullNode

医学图像重建算法

平行光束算法

2.6.4 FBP (先滤波后反投影) 算法的推导

我们首先给出二维傅里叶变换在极坐标系下的表达式

$$f(x,y)=\int_0^{2\pi}\int_0^{\infty}F_{polar}(\omega,\theta)e^{2\pi i\omega(x\cos\theta+y\sin\theta)}\omega d\omega d\theta \text{ 。}$$

因为 $F_{polar}(\omega,\theta)=F_{polar}(-\omega,\theta+\pi)$，所以

$$f(x,y)=\int_0^{\pi}\int_{-\infty}^{\infty}F_{polar}(\omega,\theta)|\omega|e^{2\pi i\omega(x\cos\theta+y\sin\theta)}d\omega d\theta \text{ 。}$$

根据中心切片定理，我们可以用 P 来代替 F：

$$f(x,y)=\int_0^{\pi}\int_{-\infty}^{\infty}P(\omega,\theta)|\omega|e^{2\pi i\omega(x\cos\theta+y\sin\theta)}d\omega d\theta \text{ 。}$$

我们意识到 $|\omega|$ 是斜坡率波器的传递函数，令 $Q(\omega,\theta)=|\omega|P(\omega,\theta)$，则

$$f(x,y)=\int_0^{\pi}\int_{-\infty}^{\infty}Q(\omega,\theta)e^{2\pi i\omega(x\cos\theta+y\sin\theta)}d\omega d\theta \text{ 。}$$

利用一维傅里叶反变换并记 Q 的反变换为 q，我们最后得到

$$f(x,y)=\int_0^{\pi}q(x\cos\theta+y\sin\theta,\theta)d\theta \text{ ，}$$

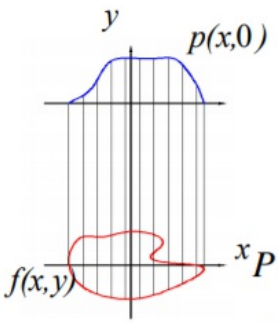
或

$$f(x,y)=\int_0^{\pi}q(s,\theta)|_{s=x\cos\theta+y\sin\theta}\,d\theta \text{ 。}$$

这正是 $q(s,\theta)$ 的反投影 (见第1.5节)。

中心切片定理

Special example prove



1. 平行于y轴投影

$$p(x,0)=\int_{-\infty}^{+\infty} f(x,y)dy \qquad (1)$$

2. 投影函数一维傅里叶变换

$$P(u)=\int_{-\infty}^{+\infty} p(x,0)e^{-j2\pi ux}dx=\int_{-\infty}^{+\infty}\int_{-\infty}^{+\infty} f(x,y)e^{-j2\pi ux}dxdy \qquad (2)$$

3. 密度函数二维傅里叶变换

$$F(u,v)=\int_{-\infty}^{+\infty}\int_{-\infty}^{+\infty} f(x,y)e^{-j2\pi(ux+vy)}dxdy \qquad (3)$$

4. 在v=0时，傅里叶变换表示

$$F(u,v)\big|_{v=0}=\int_{-\infty}^{+\infty}\int_{-\infty}^{+\infty} f(x,y)e^{-j2\pi ux}dxdy \qquad (4)$$

这表明，
一个物体0°投影的傅里叶变换与该物体二维傅里叶变换中v=0的直线相同。

5. 式（2）与（4）相等，证明完毕

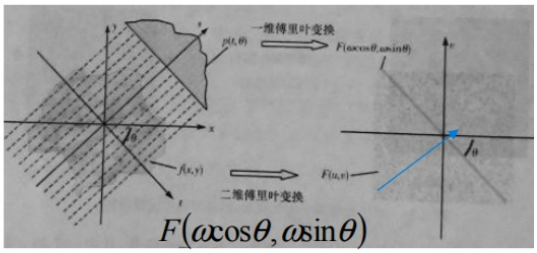
3. 对投影函数中的变量t进行傅里叶变换

$$P(\omega,\theta)=\int_{-\infty}^{+\infty}\int_{-\infty}^{+\infty} f'(t,s)dse^{-j2\pi\omega t}dt \qquad (3)$$

4. 对（3）式右面进行坐标变换x，y

$$dsdt=Jdxdy=\begin{vmatrix}\partial t/\partial x & \partial s/\partial x\\ \partial t/\partial y & \partial s/\partial y\end{vmatrix}dxdy=dxdy$$

雅克比转换



傅里叶切片定理：
物体f(x,y)平行投影的傅里叶变换，是物体二维傅里叶变换的一个切片（直线），切片是在与投影相同的角度获得的。

$$P(\omega,\theta)=\int_{-\infty}^{+\infty}\int_{-\infty}^{+\infty} f(x,y)e^{-j2\pi\omega(x\cos\theta+y\sin\theta)}dxdy \qquad (4)$$

5. f(x,y)二维傅里叶变换

$$F(u,v)=\int_{-\infty}^{+\infty}\int_{-\infty}^{+\infty} f(x,y)e^{-j2\pi(ux+vy)}dxdy \qquad (5)$$

6. 比较公式（4）和（5），令 $u=\omega\cos\theta, v=\omega\sin\theta$

$$F(\omega\cos\theta,\omega\sin\theta)=P(\omega,\theta) \qquad (6)$$

平行光束算法到扇形光束算法

其本质就是从直角坐标系变换到极坐标系

在完成了从平行光束数据 $p(s,\theta)$ 到扇形束数据 $g(\gamma,\beta)$ 的替换，旧变量 s 和 θ 到新变量 γ 和 β 的替换，及加入一个雅可比因子 $J(\gamma,\beta)$，一个崭新的扇形束图像重建算法就诞生了 (图 3.5)!

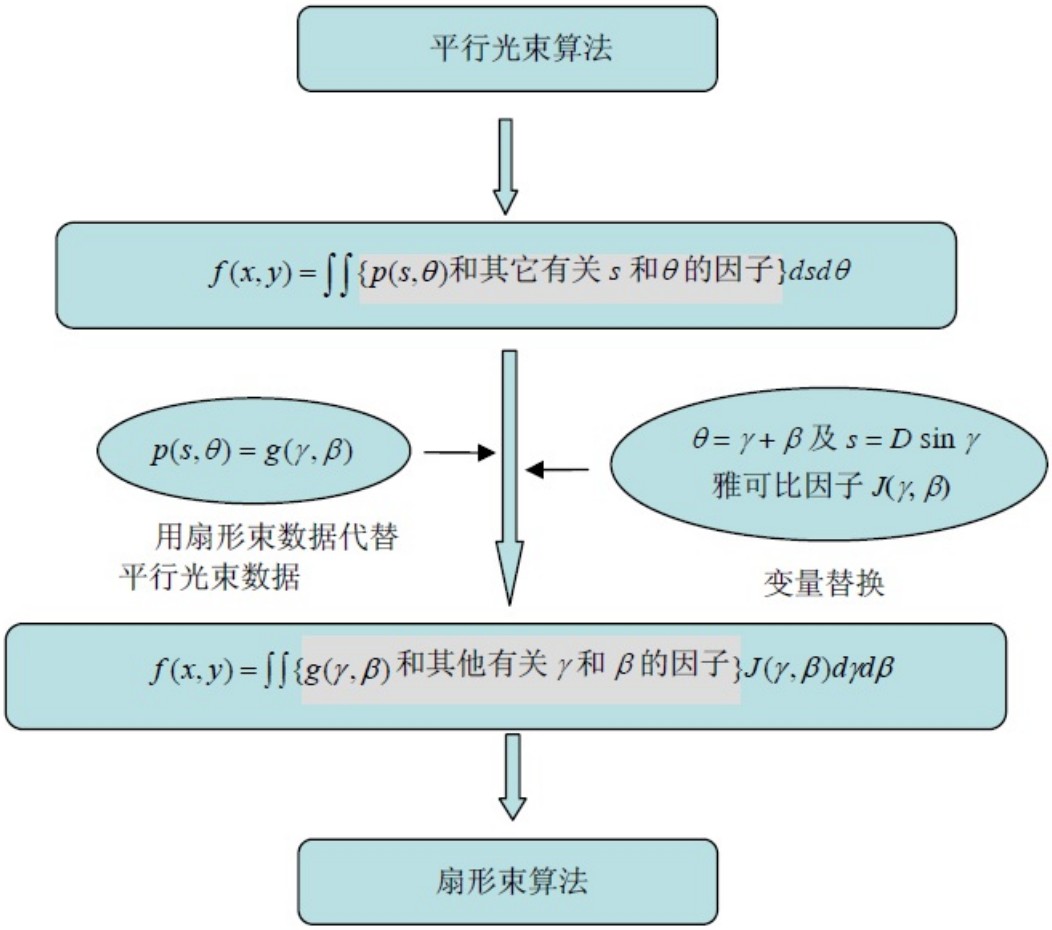


图 3.5 从平行光束图像重建算法到扇形束图像重建算法的推导过程。

投影

1.5.1 投影

设 $f(x, y)$ 为 x-y 平面上定义的密度函数，其投影函数(即射线和，线积分，及 拉东变换) $p(s, \theta)$ 有下面不同的等价表达式：

$$p(s,\theta)=\int_{-\infty}^{\infty}\int_{-\infty}^{\infty}f(x,y)\delta(x\cos\theta+y\sin\theta-s)dxdy\ ,$$

$$p(s,\theta)=\int_{-\infty}^{\infty}\int_{-\infty}^{\infty}f(x,y)\delta(\bar{x}\cdot\bar{\theta}-s)dxdy\ ,$$

$$p(s,\theta)=\int_{-\infty}^{\infty}f(s\cos\theta-t\sin\theta,s\sin\theta+t\cos\theta)dt\ ,$$

$$p(s,\theta)=\int_{-\infty}^{\infty}f(s\bar{\theta}+t\bar{\theta}^{\perp})dt\ ,$$

$$p(s,\theta)=\int_{-\infty}^{\infty}f_{\theta}(s,t)dt\ ,$$

其中 $\bar{x}=(x,y)$， $\bar{\theta}=(\cos\theta,\sin\theta)$， $\bar{\theta}^{\perp}=(-\sin\theta,\cos\theta)$，$\delta$是狄拉克$\delta$函数，图像 f 旋转角度 θ 后记为 f_{θ}。我们假设探测器按逆时针方向绕物体旋转。这等价于探测器不动，而物体按顺时针做旋转. 有关的坐标系如图1.12所示。

狄拉克函数

$$\int_{-\infty}^{\infty}\delta(ax)f(x)dx=\frac{1}{|a|}f(0)\,,$$

$$\int_{-\infty}^{\infty}\delta^{(n)}(x)f(x)dx=(-1)^n f^{(n)}(0)\ \text{[第 } n \text{ 阶导数]},$$

$\delta(g(x))f(x)=\sum_n\frac{1}{|g'(\lambda_n)|}\delta(x-\lambda_n)$ ，其中 λ_n 为 $g(x)$ 的零点。

狄拉克δ函数在二维和三维的情形的定义分别是， $\delta(\bar{x})=\delta(x)\delta(y)$ 和 $\delta(\bar{x})=\delta(x)\delta(y)\delta(z)$ 。这时，在上面的最后一个性质中， $|g'|$ 要分别被 $|grad(g)|=\sqrt{\left(\frac{\partial g}{\partial x}\right)^2+\left(\frac{\partial g}{\partial y}\right)^2}$ 和 $|grad(g)|=\sqrt{\left(\frac{\partial g}{\partial x}\right)^2+\left(\frac{\partial g}{\partial y}\right)^2+\left(\frac{\partial g}{\partial z}\right)^2}$ 取代。

在二维成像中，我们通常用δ函数 $\delta(\bar{x}-\bar{x}_0)$ 来表示位于 $\bar{x}=\bar{x}_0$ 的点源。函数 $f(\bar{x})=\delta(\bar{x}-\bar{x}_0)=\delta(x-x_0)\delta(y-y_0)$ 的拉东变换就是

$$p(s,\theta)=\int_{-\infty}^{\infty}\int_{-\infty}^{\infty}f(\bar{x})\delta(\bar{x}\cdot\bar{\theta}-s)d\bar{x}\,,$$

$$p(s,\theta)=\int_{-\infty}^{\infty}\int_{-\infty}^{\infty}\delta(x-x_0)\delta(y-y_0)\delta(x\cos\theta+y\sin\theta-s)dxdy\,,$$

解析法应用的就是**中心切片定理**，**迭代法一般**应用**共轭梯度法或者**拟**牛顿法。**

最优化

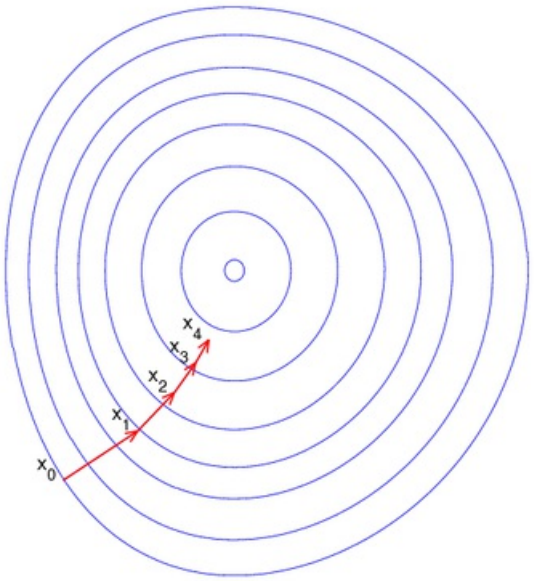
最速下降法，牛顿法，LBFGS

1.梯度下降法（Gradient Descent）

梯度下降法是最早最简单，也是最为常用的最优化方法。梯度下降法实现简单，当目标函数是凸函数时，梯度下降法的解是全局解。一般情况下，其解不保证是全局最优解，梯度下降法的速度也未必是最快的。

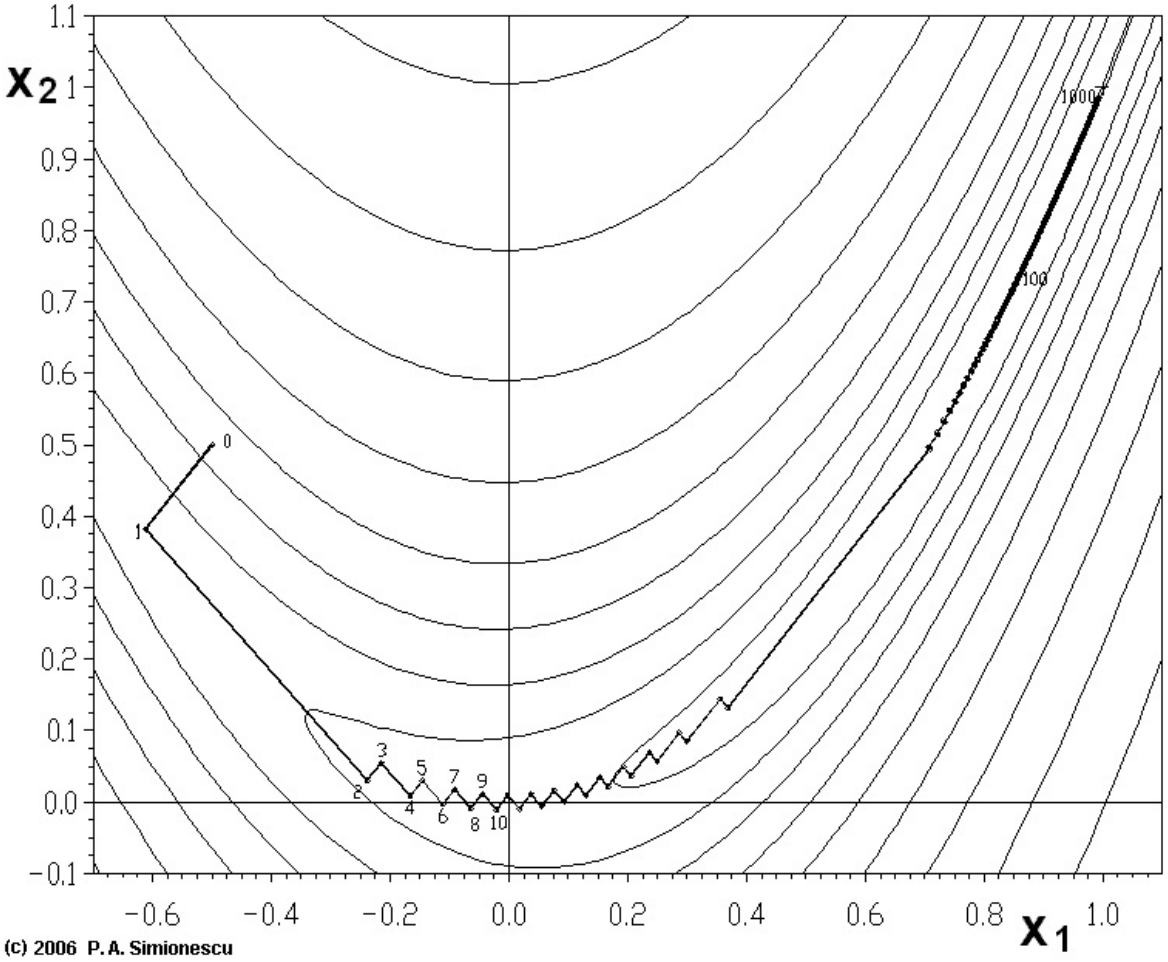
梯度下降法的优化思想是用当前位置负梯度方向作为搜索方向，因为该方向为当前位置的最快下降方向，所以也被称为是"最速下降法"。最速下降法越接近目标值，梯度趋于0，所以步长越小，前进越慢。

梯度下降法的搜索迭代示意图如下图所示：



梯度下降法的缺点：

(1)越靠近极小值的地方收敛速度越慢，如下图所示



从上图可以看出，梯度下降法在接近最优解的区域收敛速度明显变慢，利用梯度下降法求解需要很多次的迭代。

在机器学习中，基于基本的梯度下降法发展了两种梯度下降方法，分别为随机梯度下降法和批量梯度下降法。

比如对一个线性回归(Linear Logistics)模型，假设下面的h(x)是要拟合的函数，J(theta)为损失函数，theta是参数，要迭代求解的值，theta求解出来了那最终要拟合的函数h(theta)就出来了。其中m是训练集的样本个数，n是特征的个数。

$h(\theta) = \sum_{j=0}^{n} \theta_k x_j$

$J(\theta) = \frac{1}{2m} \sum_{i=0}^{m} (y^i - h_\theta(x^i))^2$
//n是特征的个数，m的训练样本的数目

1)批量梯度下降法（Batch Gradient Descent, BDG）

(1)将$J(\theta)$对θ求偏导，得到每个θ对应的梯度：

$\frac{\partial J(\theta)}{\partial \theta_j} = -\frac{1}{m} \sum_{i=1}^{m} (y^i - h_\theta(x^i)) x_j^i$
(2)由于要最小化风险函数，所以按照每个参数θ的负梯度方向来更新θ

$\theta'_j = \theta j - \frac{\partial J(\theta)}{\partial \theta_j} = \theta j + \frac{1}{m} \sum_{i=1}^{m} (y^i - h_\theta(x^i)) x_j^i$

(3)从上面公式可以注意到，它得到的是一个全局最优解，但是每迭代一步，都要用到训练集所有的数据，如果m很大，那么可想而知这种方法的迭代速度会相当的慢。所以，这就引入了另外一种方法——随机梯度下降。

对于批量梯度下降法，样本个数m，x为n维向量，一次迭代需要把m个样本全部带入计算，迭代一次计算量为m*n2。

2)随机梯度下降（Stochastic Gradient Descent, SGD）

(1)上面的风险函数可以写成如下这种形式，损失函数对应的是训练集中每个样本的粒度，而上面批量梯度下降对应的是所有的训练样本

$J(\theta) = \frac{1}{2m}\sum_{i=0}^{m}(y^i - h_\theta(x^i))^2 = \frac{1}{m}\sum_{i=1}^{m} cost(\theta,(x^i,y^i))$

$cost(\theta,(x^i,y^i)) = \frac{1}{2}(y^i - h_\theta(x^i))^2$

(2)每个样本的损失函数，对θ求偏导得到对应的梯度，来更新θ

$\theta'_j = \theta_j + (y^i - h_\theta(x^i))x^i_j$

(3)随机梯度下降是通过每个样本来迭代更新一次，如果样本量很大的情况（例如几十万），那么可能只用其中几万条或者几千条的样本，就已经将θ迭代到最优解了，对比上面的批量梯度下降，迭代一次需要用到十几万训练样本，一次迭代不可能最优，如果迭代10次的话就需要遍历训练样本10次。但是，SGD伴随的一个问题是噪音较BGD要多，使得SGD并不是每次迭代都向着整体最优化方向。

随机梯度下降每次迭代只使用一个样本，迭代一次计算量为n2，当样本个数m很大的时候，随机梯度下降迭代一次的速度要远高于批量梯度下降方法。两者的关系可以这样理解：随机梯度下降方法以损失很小的一部分精确度和增加一定数量的迭代次数为代价，换取了总体的优化效率的提升。增加的迭代次数远远小于样本的数量。

对批量梯度下降法和随机梯度下降法的总结：

批量梯度下降---最小化所有训练样本的损失函数，使得最终求解的是全局的最优解，即求解的参数是使得风险函数最小，但是对于大规模样本问题效率低下。

随机梯度下降---最小化每条样本的损失函数，虽然不是每次迭代得到的损失函数都向着全局最优方向，但是大的整体的方向是向全局最优解的，最终的结果往往是在全局最优解附近，适用于大规模训练样本情况。

由于牛顿法是基于当前位置的切线来确定下一次的位置，所以牛顿法又被很形象地称为是"切线法"。牛顿法的搜索路径（二维情况）如下图所示：

牛顿法搜索动态示例图：

牛顿法和拟牛顿法（Newton's method &Quasi-Newton methods）

1)牛顿法（Newton's method）

牛顿法是一种在实数域和复数域上近似求解方程的方法。方法使用函数f (x)的泰勒级数的前面几项来寻找方程f (x) = 0的根。牛顿法最大的特点就在于它的收敛速度很快。[1]

步骤：

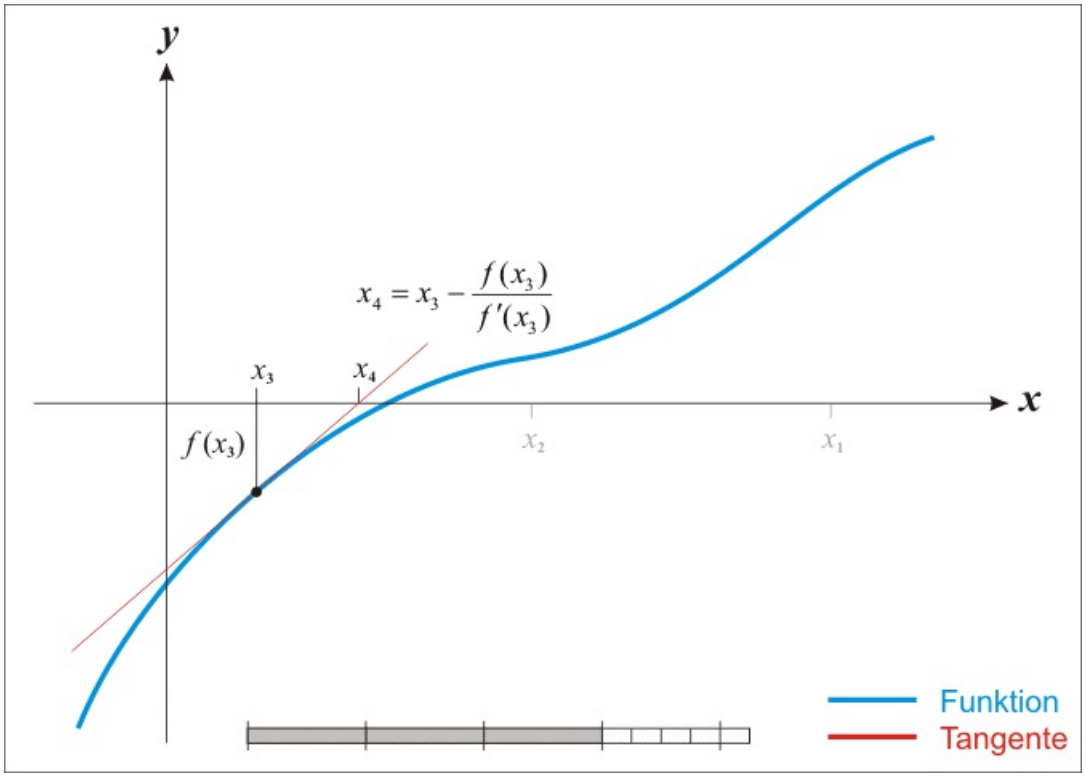
首先，选择一个接近函数f(x)零点的x_0,计算相应的计算相应的 $f(x_0)$ 和切线斜率$f'(x_0)$。然后计算穿过点$(x_0, f(x_0))$并且斜率为$f'(x_0)$的直线和x轴的交点的x轴坐标，也就是如下方程：

$x * f'(x_0) + f(x_0) - x_0 * f'(x_0) = 0$

我们将新求得的点的 x 坐标命名为x_1，通常x_1会比x_0更接近方程$f(x) = 0$的解。因此我们现在可以利用x_1开始下一轮迭代。迭代公式可化简为如下所示：

$x_{n+1} = x_n + \frac{f(x_n)}{f'(x_n)}$

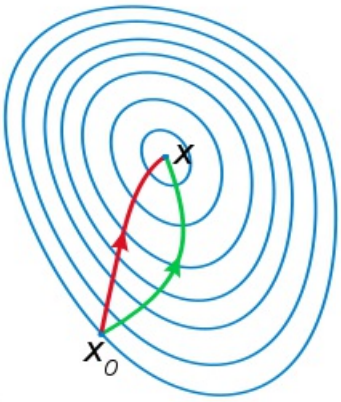
已经证明，如果$f'(x)$是连续的，并且待求的零点x是孤立的，那么在零点x周围存在一个区域，只要初始值x_0位于这个邻近区域内，那么牛顿法必定收敛。并且，如果$f'(x)$不为0, 那么牛顿法将具有平方收敛的性能. 粗略的说，这意味着每迭代一次，牛顿法结果的有效数字将增加一倍。下图为一个牛顿法执行过程的例子。



关于牛顿法和梯度下降法的效率对比：

从本质上去看，牛顿法是二阶收敛，梯度下降是一阶收敛，所以牛顿法就更快。如果更通俗地说的话，比如你想找一条最短的路径走到一个盆地的最底部，梯度下降法每次只从你当前所处位置选一个坡度最大的方向走一步，牛顿法在选择方向时，不仅会考虑坡度是否够大，还会考虑你走了一步之后，坡度是否会变得更大。所以，可以说牛顿法比梯度下降法看得更远一点，能更快地走到最底部。（牛顿法目光更加长远，所以少走弯路；相对而言，梯度下降法只考虑了局部的最优，没有全局思想。）

根据**wiki**上的解释，从几何上说，牛顿法就是用一个二次曲面去拟合你当前所处位置的局部曲面，而梯度下降法是用一个平面去拟合当前的局部曲面，通常情况下，二次曲面的拟合会比平面更好，所以牛顿法选择的下降路径会更符合真实的最优下降路径。



注：红色的牛顿法的迭代路径，绿色的是梯度下降法的迭代路径。

牛顿法的优缺点总结：

优点：二阶收敛，收敛速度快；

缺点：牛顿法是一种迭代算法，每一步都需要求解目标函数的**Hessian**矩阵的逆矩阵，计算比较复杂。

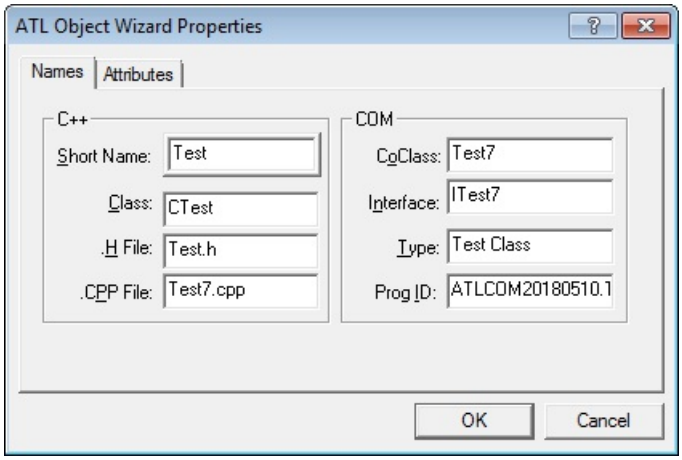
第六章 C++

第一节 基本语法

接口：即抽象类，就是包含至少一个纯虚函数的类

```
一个类里面实现多种接口Iinterface, IinterfaceB, IinterfaceC
    IE84TPTimeOutUIAck : public IUnknown
    {
    public:
        virtual HRESULT STDMETHODCALLTYPE ResponseTpTimeOutUIAck(
            /* [in] */ short nPortNo,
            /* [in] */ short nTpNo) = 0;

    };
```

New ATL object 时，选择simple object，然后属性设置如下：则可以实现一个类中有多组接口函数。



C++通过ATL来实现接口的继承。

```
// Cr3halObj
class ATL_NO_VTABLE Cr3halObj :
    public CComObjectRootEx<CComSingleThreadModel>,
    public CComCoClass<Cr3halObj, &CLSID_r3halObj>,
    public IConnectionPointContainerImpl<Cr3halObj>,
    public Ir3halObj,
    public IEmuConnectionPointImpl<Cr3halObj, &IID_IHALEvents>,
    public IEmuConnectionPointImpl<Cr3halObj, &IID_IE84Events>,
    public IEmuConnectionPointImpl<Cr3halObj, &IID_IMMIEvents>,
    public IEmuConnectionPointImpl<Cr3halObj, &IID_IE90Events>,
    public IEmuConnectionPointImpl<Cr3halObj, &IID_IE116Events>,
    public IEmuConnectionPointImpl<Cr3halObj, &IID_IFFUEvents>,
    public IEmuConnectionPointImpl<Cr3halObj, &IID_IWaferProtrusionEvent>,
    public IEmuConnectionPointImpl<Cr3halObj, &IID_ISignalTowerDevice>,
    public IEmuConnectionPointImpl<Cr3halObj, &IID_IE84HOAVBLEvents>,
    public IEmuConnectionPointImpl<Cr3halObj, &IID_IE84TPTimeOutUIAck>, //TP3 connection point
    public IR3HALDiag
{

DECLARE_EMUCLASSFACTORY_SINGLETON(Cr3halObj)
```

COM组件接口继承的实现 https://blog.csdn.net/dingbaosheng/article/details/624504

TL学习笔记03

继承

C++:

C#:

http://www.cnblogs.com/flyinthesky/archive/2008/06/18/1224774.html

不能初始化的类被叫做抽象类，它们只提供部分实现，但是另一个类可以继承它并且能创建它们
的实例。"一个包含一个或多个纯虚函数的类叫抽象类，抽象类不能被实例化，进一步一个抽象类只能通过接口和作为其它类的基类使用.

"抽象类能够被用于类，方法，属性，索引器和事件，使用abstract 在一个类声明中表示该类倾向要作为其它类的基类

成员被标示成abstract，或被包含进一个抽象类，必须被其派生类实现.

一个抽象类可以包含抽象和非抽象方法，当一个类继承于抽象类，那么这个派生类必须实现所有的的基类抽象方法。

一个抽象方法是一个没有方法体的方法。

1. virtual修饰的方法必须有实现（哪怕是仅仅添加一对大括号),而abstract修饰的方法一定不能实现。
2. virtual可以被子类重写，而abstract必须被子类重写，
3. 如果类成员被abstract修饰，则该类前必须添加abstract，因为只有抽象类才可以有抽象方法。
4. 无法创建abstract类的实例，只能被继承无法实例化，
5. C#中如果要在子类中重写方法，必须在父类方法前加virtual，在子类方法前添加override,这样就避免了程序员在子类中不小心重写了父类方法。
6. abstract方法必须重写，virtual方法必须有实现(即便它是在abstract类中定义的方法).

mRet = TestHalfWafer(token, ref nSeq, true); //C# ref

模板类与模板函数

Why can templates only be implemented in the header file?

Clarification: header files are not the _only _portable solution. But they are the most convenient portable solution Error LNK2019 unresolved external symbol "public: int __thiscall Algo<int>::LongestIncreaseSubsequence(int * const,int)" (?LongestIncreaseSubsequence@?$Algo@H@@QAEHQAHH@Z) referenced in function _main Algo C:\Data\ShareFolder\Group\Tim\Code\Algo\Algo\Main.obj

当模板声明在头文件，实现在cpp中，一般会出现上面的问题，原因是
Algo<int> Alg实例化的时候，编译器会去创建一个int型的Algo的新类，以及LongestIncreaseSubsequence(T arr[], int arrSize)方法，因此编译器会去找这个方法的实现，如果没有找到，自然会报错。因此有如下两个方法

1. 在头文件中实现
2. 在头文件中声明，再在cpp中显示实例化(explicit instantiations) template class Algo<int>;并实现模板函数

方式1：实现在头文件里

```
#pragma once
#include <algorithm>
using namespace std;

template<class T>
class Algo
{
public:
    Algo() {};
    ~Algo() {};
public:
    int LongestIncreaseSubsequence(T arr[], int arrSize)
    {
        int *maxLen = new int[arrSize]();
        for (int i = 0; i < arrSize; i++)
        {
            maxLen[i] = 1;
        }
        for (int j = 1; j < arrSize; j++)
        {
            int maxLenJ = 1;
            for (int i = 0; i < j; i++)
            {
                if (arr[i] < arr[j])
                {
                    maxLenJ = maxLen[i] + 1;
                    maxLen[j] = std::max({ maxLen[j] ,maxLenJ });
                }
            }
        }
        return maxLen[arrSize - 1];
    }
};
```

```
//调用函数
#include "stdafx.h"
#include "Algo.h"
#include <iostream>
using namespace std;

int main()
{
    int param[6] = {5,3,4,8,6,7};
    int arrSize = sizeof(param) / sizeof(int);
    Algo<int> Alg;
    int maxLength = Alg.LongestIncreaseSubsequence(param, arrSize);
    cout << "Max length: " << maxLength << endl;
    return 0;
}
```

方式二：在头文件中声明，CPP中定义并对每种实例进行声明

```
Algo.h
#pragma once
#include <algorithm>
using namespace std;
```

```
template<class T>
class Algo
{
public:
    Algo() {};
    ~Algo() {};
public:
    int LongestIncreaseSubsequence(T arr[], int arrSize);
}
```

```
// Algo.cpp : Defines the entry point for the console application.

#include "stdafx.h"
#include "Algo.h"
#include <algorithm>
using namespace std;

template class Algo<int>; //必须先声明，才可以用
template class Algo<double>;
template class Algo<char>;
template<class T>
int Algo<T>::LongestIncreaseSubsequence(T arr[], int arrSize)
{
    int *maxLen = new int[arrSize]();
    for (int i = 0; i < arrSize; i++)
    {
        maxLen[i] = 1;
    }
    for (int j = 1; j < arrSize; j++)
    {
        int maxLenJ = 1;
        for (int i = 0; i < j; i++)
        {
            if (arr[i] < arr[j])
            {
                maxLenJ = maxLen[i] + 1;
                maxLen[j] = std::max({ maxLen[j] ,maxLenJ });
            }
        }
    }
    return maxLen[arrSize - 1];
}
```

```
If all you want to know is how to fix this situation, read the next two FAQs.
But in order to understand why things are the way they are, first accept these facts:

A template is not a class or a function. A template is a “pattern” that the compiler uses to generate a family of classes or functions.
In order for the compiler to generate the code, it must see both the template definition (not just declaration) and the specific types/whatever use
d to “fill in” the template. For example, if you’re trying to use a Foo<int>, the compiler must see both the Foo template and the fact that you’re
trying to make a specific Foo<int>.
Your compiler probably doesn’t remember the details of one .cpp file while it is compiling another .cpp file. It could, but most do not and if you
are reading this FAQ, it almost definitely does not. BTW this is called the “separate compilation model.”
Now based on those facts, here’s an example that shows why things are the way they are. Suppose you have a template Foo defined like this:

template<typename T>
class Foo {
public:
  Foo();
  void someMethod(T x);
private:
  T x;
};
Along with similar definitions for the member functions:

template<typename T>
Foo<T>::Foo()
{
  // ...
}
template<typename T>
void Foo<T>::someMethod(T x)
{
  // ...
}
Now suppose you have some code in file Bar.cpp that uses Foo<int>:

// Bar.cpp
void blah_blah_blah()
{
  // ...
```

```
  Foo<int> f;
  f.someMethod(5);
  // ...
}
Clearly somebody somewhere is going to have to use the “pattern” for the constructor definition and for the someMethod() definition and instantiate
 those when T is actually int.
But if you had put the definition of the constructor and someMethod() into file Foo.cpp, the compiler would see the template code when it compiled
Foo.cpp and it would see Foo<int> when it compiled Bar.cpp,
but there would never be a time when it saw both the template code and Foo<int>. So by rule #2 above, it co
```

```
uld never generate the code for  Foo<int>::someMethod().
A note to the experts: I have obviously made several simplifications above. This was intentional so please don’t complain too loudly. If you know t
he difference between a .cpp file and a compilation unit, the difference between a class template and a template class, and the fact that templates
 really aren’t just glorified macros, then don’t complain: this particular question/answer wasn’t aimed at you to begin with. I simplified things s
o newbies would “get it,” even if doing so offends some experts.
```

const对象只能访问**const**成员函数。因为**const**对象表示其不可改变，而非**const**成员函数可能在内部改变了对象，所以不能调用。

而非**const**对象既能访问**const**成员函数，也能访问非**const**成员函数，因为非**const**对象表示其可以改变。

```
#ifndef _MATRIX_H
#define _MATRIX_H
#include <iostream>
#include <algorithm>
using namespace std;

template<class T>
class CMatrix
{
public:
    CMatrix() {};
    CMatrix(int rows, int columns);
    CMatrix(const CMatrix&);
    ~CMatrix() {};
public:
    void Set(int row, int column, T val);
    T Get(int row, int column) const;
    CMatrix operator +(const CMatrix &mat2);
    CMatrix operator -(const CMatrix &mat2);
    CMatrix operator *(const CMatrix &mat2);
    CMatrix operator *(const T div);
    CMatrix operator /(const T div);
    CMatrix I(int n);
    int Rows() const{ return mRows; }
    int Columns() const{ return mColumns; }
    int Length() { return mRows*mColumns; }
    friend ostream &operator<<(ostream &os, const CMatrix &mat)
    {
        int row = mat.Rows();
        int col = mat.Columns();
        //mat是const，因此只能访问const成员函数

        for (int i = 0; i < row; i++)
        {
            for (int j = 0; j < col; j++)
            {
                os << fixed << mat.Get(i, j) << '\t';
            }
            if (i < row)
            {
                os << "\n";
            }
        }
        os << "\n";
        return os;
    };

private:
    int mRows;
    int mColumns;
    T* startElement;

};
#endif
```

COM连接点

COM连接点 - 最简单的例子

```
AtlAdvise(spCar, sinkptr, __uuidof(_IMyCarEvents), &cookies);
sinkptr指向了一个CComObject<CSink>
CComObject<CSink>* sinkptr = nullptr;
CComObject<CSink>::CreateInstance(&sinkptr);
```

换句话说，一旦将sink对象成功挂载到了COM对象，那么sink对象的生命周期就由对应的COM(spCar)对象来管理。

一旦挂载成功，sink对象就托管给对应的COM对象了，如果对应的COM对象析构了，那么所有它管理的sink对象也就释放了。

```
CLR是什么？
```

COM Interop(互操作)

引言

- 平台调用
- **C++ Interop(互操作)**
- **COM Interop(互操作)**

一、引言

这个系列是在C#基础知识中遗留下来的一个系列的，因为在C# 4.0中的一个新特性就是对COM互操作改进，然而COM互操作性却是.NET平台下其中一种互操作技术，为了帮助大家更好的了解.NET平台下的互操作技术，所以才有了这个系列。然而有些朋友们可能会有这样的疑问——“为什么我们需要掌握互操作技术的呢？”对于这个问题的解释就是——掌握了.NET平台下的互操作性技术可以帮助我们在.NET中调用非托管的dll和COM组件。.NET是建立在操作系统的之上的一个开发框架，其中.NET 类库中的类也是对Windows API的抽象封装，然而.NET类库不可能对所有Windows API进行封装，当.NET中没有实现某个功能的类，然而该功能在Windows API被实现了，此时我们完全没必要去自己在.NET中自定义个类，这时候就可以调用Windows API 中的函数来实现，此时就涉及到托管代码与非托管代码的交互，此时就需要使用到互操作性的技术来实现托管代码和非托管代码更好的交互。.NET 平台下提供了3种互操作性的技术：

1. Platform Invoke(P/Invoke)，即平台调用,主要用于调用C库函数和Windows API
2. C++ Introp，主要用于Managed C++(托管C++)中调用C++类库
3. COM Interop，主要用于在.NET中调用COM组件和在COM中使用.NET程序集。

下面就对这3种技术分别介绍下。

二、平台调用

使用平台调用的技术可以在托管代码中调用动态链接库(Dll)中实现的非托管函数，如Win32 Dll和C/C++ 创建的dll。看到这里，有些朋友们应该会有疑问——在怎样的场合我们可以使用平台调用技术来调用动态链接库中的非托管函数呢？

这个问题就如前面引言中说讲到的一样，当在开发过程中，.NET类库中没有提供相关API然而Win32 API 中提供了相关的函数实现时，此时就可以考虑使用平台调用的技术在.NET开发的应用程序中调用Win32 API中的函数；

然而还有一个使用场景就是——由于托管代码的效率不如非托管代码，为了提高效率，此时也可以考虑托管代码中调用C库函数。

2.1 在托管代码中通过平台调用来调用非托管代码的步骤

(1). 获得非托管函数的信息，即dll的名称，需要调用的非托管函数名等信息

(2). 在托管代码中对非托管函数进行声明，并且附加平台调用所需要属性

(3). 在托管代码中直接调用第二步中声明的托管函数

2.2 平台调用的调用过程

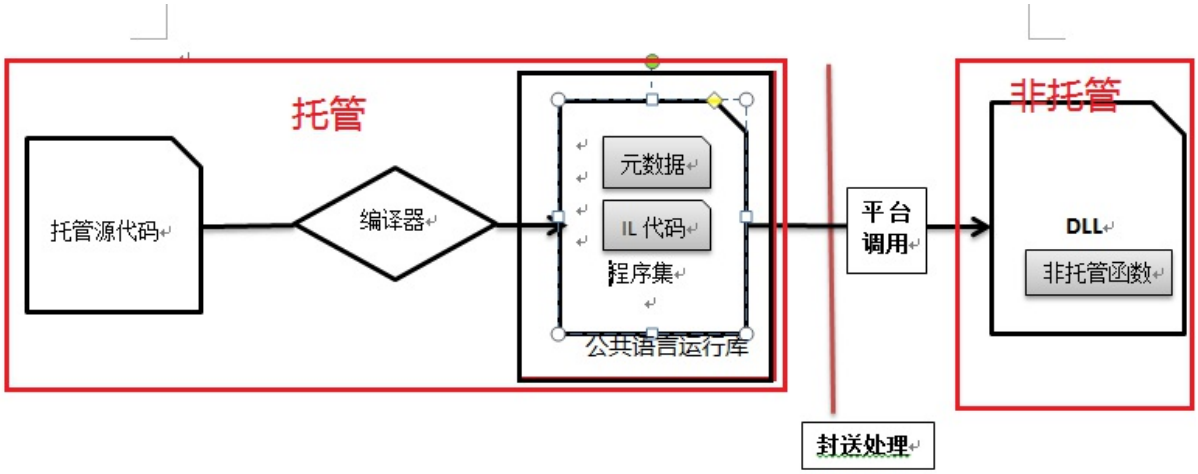
(1) 查找包含该函数的DLL，当需要调用某个函数时，当然第一步就需要知道包含该函数的DLL的位置，所以平台调用的第一步也就是查找DLL，其实在托管代码中调用非托管代码的调用过程可以想象成叫某个人做事情，首先我们要找到那个人在哪里(即查找函数的**DLL**过程)，找到那个人之后需要把要做的事情告诉他(相当于加载**DLL**到内存中和传入参数)，最后让他去完成需要完成的事情(相当于让非托管函数去执行任务)。

(2) 将找到的DLL加载到内存中。

(3) 查找函数在内存中的地址并把其参数推入堆栈，来封送所需的数据。CLR只会在第一次调用函数时，才会去查找和加载DLL，并查找函数在内存中的地址。当函数被调用过一次之后，CLR会将函数的地址缓存起来，CLR这种机制可以提高平台调用的效率。在应用程序域被卸载之前，找到的DLL都一直存在于内存中。

(4) 执行非托管函数。

平台调用的过程可以通过下图更好地理解：



三、C++ Interop

第二部分主要向大家介绍了第一种互操作性技术，然后我们也可以使用C++ Interop技术来实现与非托管代码进行交互。然而C++ Interop 方式有一个与平台调用不一样的地方，就是C++ Interop 允许托管代码和非托管代码存在于一个程序集中，甚至同一个文件中。C++ Interop 是在源代码上直接链接和编译非托管代码来实现与非托管代码进行互操作的，而平台调用是加载编译后生成的非托管DLL并查找函数的入口地址来实现与非托管函数进行互操作的。**C++ Interop**使用托管**C++**来包装非托管**C++**代码，然后编译生成程序集，然后再托管代码中引用该程序集，从而来实现与非托管代码的互操作。关于具体的使用和与平台调用的比较，这里就不多介绍，我将会在后面的专题中具体介绍。

COM(Component Object Model. 组件对象模型)是微软之前推荐的一种开发技术，由于微软过去十多年里开发了大量的COM组件，

然而不可能再使用.Net技术重写这些COM组件实现的功能，所以为了解决在.Net中的托管代码能够调用COM组件中问题，.NET平台下提供了COM Interop，即COM互操作技术，COM interop不仅支持在托管代码中使用COM组件，而且支持向COM组件中使用.NET程序集。

1.在.NET中使用COM组件

在.NET中使用COM对象，主要有三种方法：

1. 使用TlbImpl工具为COM组件创建一个互操作程序集来绑定早期的COM对象，这样就可以在程序中添加互操作程序集来调用COM对象
2. 通过反射来后期绑定COM对象
3. 通过P/Invoke创建COM对象或使用C++ Interop为COM对象编写包装类

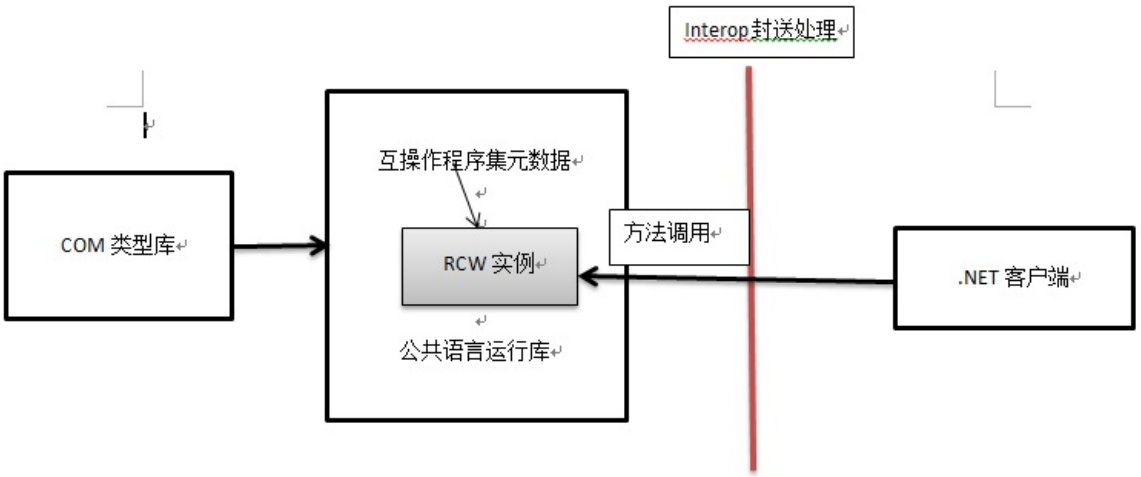
但是我们经常使用的都是方法一，下面介绍下使用方法一在.NET 中使用COM对象的步骤：

1. 找到要使用的COM 组件并注册它。使用 regsvr32.exe 注册或注销 COM DLL。
2. 在项目中添加对 COM 组件或类型库的引用。

添加引用时，**Visual Studio** 会用到**TlbImp.exe**（类型库导入程序），**TlbImp.exe**程序将生成一个 **.NET Framework** 互操作程序集。该程序集又称为运行时可调用包装 **(RCW)**，其中包含了包装**COM**组件中的类和接口。**Visual Studio** 将生成组件的引用添加至项目。

1. 创建RCW中类的实例，这样就可以使用托管对象一样来使用COM对象。

下面通过一个图更好地说明在.NET中使用COM组件的过程：

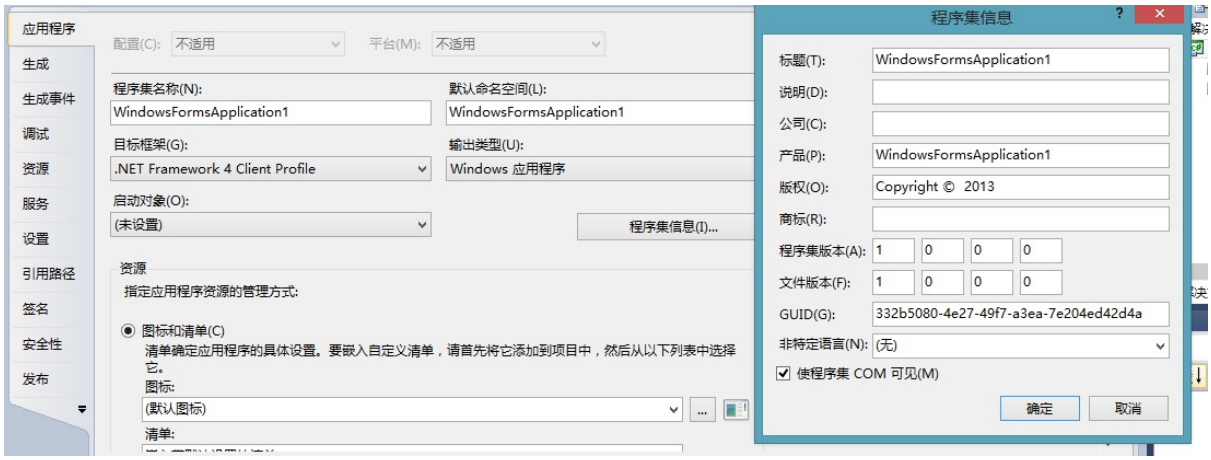


在COM中使用.NET程序集

.NET 公共语言运行时通过COM可调用包装（COM Callable Wrapper,即CCW）来完成与COM类型库的交互。CCW可以使COM客户端认为是在与普通的COM类型交互，同时使.NET组件认为它正在与托管应用程序交互。在这里CCW是非托管COM客户端与托管对象之间的一个代理。CCW既可以维护托管对象的生命周期，也负责数据类型在COM和.NET之间的相互转换。实现在COM使用.NET 类型的基本步骤如：

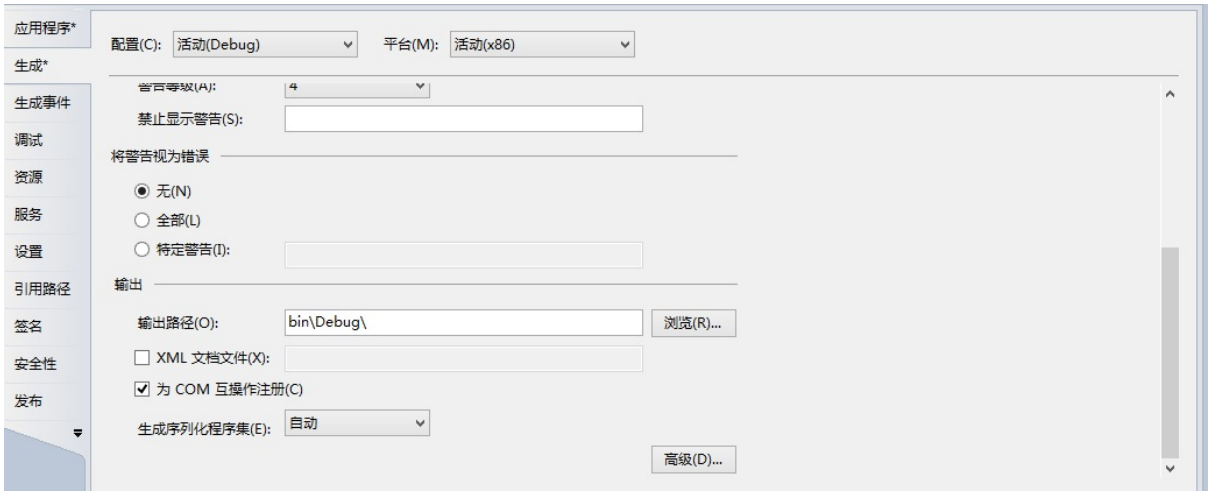
1. 在C#项目中添加互操作特性

   可以修改C#项目属性使程序集对COM可见。右键解决方案选择属性，在“应用程序标签”中选择“程序集信息”按钮，在弹出的对话框中选择 “使程序集COM可见” 选项，如下图所示：



1. 生成COM类型库并对它进行注册以供COM客户端使用

在“生成”标签中，选中 “为COM互操作注册”选项，如下图：



勾选“为COM互操作注册”选项后，Visual Studio会调用类型库导出工具(Tlbexp.exe)为.NET程序集生成COM类型库再使用程序集注册工具(Regasm.exe)来完成对.NET程序集和生成的COM类型库进行注册，这样COM客户端可以使用CCW服务来对.NET对象进行调用了。

总结

介绍到这里，本专题的内容就结束，本专题主要对.NET 提供的互操作的技术做了一个总的概括，在后面的专题中将会对具体的技术进行详细的介绍和给出一些简单的使用例子。

驱动

```
WINBASEAPI
BOOL
WINAPI
DeviceIoControl(
    HANDLE hDevice,
    DWORD dwIoControlCode,
    LPVOID lpInBuffer,
    DWORD nInBufferSize,
    LPVOID lpOutBuffer,
    DWORD nOutBufferSize,
    LPDWORD lpBytesReturned,
    LPOVERLAPPED lpOverlapped
    );
是WinAPI, 负责与硬件打交道, 收发数据, 然后在我们的驱动程序DispatchControl函数中, 去解析DeviceIoControl传送的数据
```

```
// Control.cpp -- IOCTL handlers for usb42 driver
// Copyright (C) 1999 by Walter Oney
// All rights reserved

#include "stddcls.h"
#include "driver.h"
#include "ioctls.h"

///////////////////////////////////////////////////////////////////////////////

#pragma PAGEDCODE

NTSTATUS DispatchControl(PDEVICE_OBJECT fdo, PIRP Irp)
    {                           // DispatchControl
    PAGED_CODE();
    PDEVICE_EXTENSION pdx = (PDEVICE_EXTENSION) fdo->DeviceExtension;

    NTSTATUS status = IoAcquireRemoveLock(&pdx->RemoveLock, Irp);
    if (!NT_SUCCESS(status))
        return CompleteRequest(Irp, status, 0);
    ULONG info = 0;

    PIO_STACK_LOCATION stack = IoGetCurrentIrpStackLocation(Irp);
    ULONG cbin = stack->Parameters.DeviceIoControl.InputBufferLength;
    ULONG cbout = stack->Parameters.DeviceIoControl.OutputBufferLength;
    ULONG code = stack->Parameters.DeviceIoControl.IoControlCode;

    switch (code)
        {                       // process request

    case IOCTL_USB42_READ:
        {                       // IOCTL_USB42_READ
        if (cbout != 64)
            {
            status = STATUS_INVALID_PARAMETER;
            break;
            }

        URB urb;
        UsbBuildInterruptOrBulkTransferRequest(&urb, sizeof(_URB_BULK_OR_INTERRUPT_TRANSFER),
            pdx->hpipe, Irp->AssociatedIrp.SystemBuffer, NULL, cbout, USBD_TRANSFER_DIRECTION_IN | USBD_SHORT_TRANSFER_OK, NULL);
        status = SendAwaitUrb(fdo, &urb);
        if (!NT_SUCCESS(status))
            KdPrint(("USB42 - Error %X (USBD status code %X) trying to read endpoint\n", status, urb.UrbHeader.Status));
        else
            info = cbout;
        break;
        }                       // IOCTL_USB42_READ

    default:
        status = STATUS_INVALID_DEVICE_REQUEST;
        break;

        }                       // process request

    IoReleaseRemoveLock(&pdx->RemoveLock, Irp);
    return CompleteRequest(Irp, status, info);
    }                           // DispatchControl
```

第七章 CSharp

这一章主要涉及C#的一些基本知识，比如delegate,Lamda 表达式，Event,反射

第二节 C#基本知识

Linq

```
Linq 语句
q = from sl in m_cassette.slot_descs
     where sl.slot_seq == i
     orderby sl.recipe_seq ascending
     select sl;
```

委托

```
C#委托
Action<int, string, string> dr = this.OnFinishedLoadingWaferInternal;
    this.Invoke(dr, new object[] { slot, waferID, lotID });

http://www.cnblogs.com/akwwl/p/3232679.html
Delegate至少0个参数，至多32个参数，可以无返回值，也可以指定返回值类型
Func可以接受0个至16个传入参数，必须具有返回值
Action可以接受0个至16个传入参数，无返回值
Predicate只能接受一个传入参数，返回值为bool类型
```

Mutex

```
 Mutex的使用
 // Summary:
        //     Initializes a new instance of the System.Threading.Mutex class with a Boolean
        //     value that indicates whether the calling thread should have initial ownership
        //     of the mutex, a string that is the name of the mutex, and a Boolean value that,
        //     when the method returns, indicates whether the calling thread was granted initial
        //     ownership of the mutex.
        //
        // Parameters:
        //   initiallyOwned:
        //     true to give the calling thread initial ownership of the named system mutex if
        //     the named system mutex is created as a result of this call; otherwise, false.
        //
        //   name:
        //     The name of the System.Threading.Mutex. If the value is null, the System.Threading.Mutex
        //     is unnamed.
        //
        //   createdNew:
        //     When this method returns, contains a Boolean that is true if a local mutex was
        //     created (that is, if name is null or an empty string) or if the specified named
        //     system mutex was created; false if the specified named system mutex already existed.
        //     This parameter is passed uninitialized.
        //
        // Exceptions:
        //   T:System.UnauthorizedAccessException:
        //     The named mutex exists and has access control security, but the user does not
        //     have System.Security.AccessControl.MutexRights.FullControl.
        //
        //   T:System.IO.IOException:
        //     A Win32 error occurred.
        //
        //   T:System.Threading.WaitHandleCannotBeOpenedException:
        //     The named mutex cannot be created, perhaps because a wait handle of a different
        //     type has the same name.
        //
        //   T:System.ArgumentException:
        //     name is longer than 260 characters.
        [ReliabilityContract(Consistency.WillNotCorruptState, Cer.MayFail)]
        [SecurityCritical]
        public Mutex(bool initiallyOwned, string name, out bool createdNew);
        var mutex = new Mutex(false, RS7.IF.CommonFunction.CommonString.MutexName, out createdNew);
```

as

```
as 的用法
IModuleCommonInterface iMagnet = MagnetSystem.GetInstance as IModuleCommonInterface;
is : 检查一个对象是否兼容于其他指定的类型,并返回一个Bool值,永远不会抛出异常
object o = new object();
        if (o is Label)
         {
             Label lb = (Label)o;
             Response.Write("类型转换成功");
         }
         else
         {
             Response.Write("类型转换失败");
         }
as:与强制类型转换是一样的,但是永远不会抛出异常,即如果转换不成功,会返回null
object o = new object();
         Label lb = o as Label;
         if (lb == null)
         {
             Response.Write("类型转换失败");
         }
         else
         {
             Response.Write("类型转换成功");
         }
```

Event

```
Event
public static event EventHandler<NexusWaferLocationEventArgs> WaferLocationEvent
        {
            add { m_waferLocationEvent += value; }
            remove { m_waferLocationEvent -= value; }
        }
```

Lamda

```
UserControl是C#在做Windows窗体应用程序时，经常用到的一个控件。他和Form一样，是一个展示型的控件。
本文介绍下Usercontrol在开发中常用的一些属性和方法。
```

```
C# 用Linq的方式实现对Xml文件的基本操作(创建xml文件、增删改查xml文件节点信息)
https://www.cnblogs.com/mingmingruyuedlut/archive/2011/01/27/1946239.html
https://blog.csdn.net/songyi160/article/details/50824274
```

接口

```
接口使用interface 关键字进行定义，可由方法、属性、事件、索引器或这四种成员类型的任意组合构成。
接口的特性：
1.接口类似于抽象基类，不能直接实例化接口；接口中的方法都是抽象方法，实现接口的任何非抽象类型都必须实现接口的所有成员：
当显式实现该接口的成员时，实现的成员不能通过类实例访问，只能通过接口实例访问。
当隐式实现该接口的成员时，实现的成员可以通过类实例访问，也可以通过接口实例访问，但是实现的成员必须是公有的。
2.接口不能包含常量、字段、运算符、实例构造函数、析构函数或类型、不能包含静态成员。
3.接口成员是自动公开的，且不能包含任何访问修饰符。
4.接口自身可从多个接口继承，类和结构可继承多个接口，但接口不能继承类。
```

Lamda 表达式

```
To create a lambda expression, you specify input parameters (if any) on the left side of the lambda operator =>,
and you put the expression or statement block on the other side. For example, the lambda expression x => x * x
specifies a parameter that's named x and returns the value of x squared. You can assign this expression to a
delegate type, as the following example shows:
delegate int del(int i);
static void Main(string[] args)
```

```
{
    del myDelegate = x => x * x;
    int j = myDelegate(5); //j = 25
}
public bool FloatTopLayer
{
    get => ForwardParameters.GetParameterValue<bool>(CalculationParameters.Names.JxRA_FLOAT_TOP_LAYER, true);
    set => ForwardParameters.SetParameterValue<bool>(CalculationParameters.Names.JxRA_FLOAT_TOP_LAYER, value);
}
```

Default关键字

```
在泛型类和泛型方法中产生的一个问题是，在预先未知以下情况时，如何将默认值分配给参数化类型 T：
T 是引用类型还是值类型。
如果 T 为值类型，则它是数值还是结构。
给定参数化类型 T 的一个变量 t，只有当 T 为引用类型时，语句 t = null 才有效；只有当 T 为数值类型而不是结构时，
语句 t = 0 才能正常使用。 解决方案是使用 default 关键字，此关键字对于引用类型会返回 null，对于数值类型会返回零。
对于结构，此关键字将返回初始化为零或 null 的每个结构成员，具体取决于这些结构是值类型还是引用类型。
对于可以为 null 的值类型，默认返回 System.Nullable(Of T)，它像任何结构一样初始化
public class GenericList<T> {
    private class Node {
        public Node next;
        public T Data;
    }
    private Node head;
    public T GetFirst() {
        T temp = default(T);
        if(head != null) {
            temp = head.data;
        }
        return temp
    }
}
```

为控件添加权限

```
1.让需要设置权限的控件继承一个接口IUserManagement，这个接口就只有一个方法OnUpdateUserChange，
这个方法里面处理不同用户的权限。
public void OnUpdateUserChange(UserLevel userLevel)
        {
            if (userLevel == UserLevel.USER_ADMIN)
            {
                btAdd.Enabled = true;
                btEdit.Enabled = true;
                btDelete.Enabled = true;
            }
            else if (userLevel == UserLevel.USER_ENGINEER)
            {
                btAdd.Enabled = true;
                btEdit.Enabled = true;
                btDelete.Enabled = true;
            }
            else
            {
                btAdd.Enabled = false;
                btEdit.Enabled = false;
                btDelete.Enabled = false;
            }
        }

private List<IUserManagement> m_lsUser = new List<IUserManagement>();
//User Management 把不同控件添加到m_lsUser
m_iuserManageContainer.AddModule(m_treeView);
m_iuserManageContainer.AddModule(m_sysConfigEditorCtrl);
m_iuserManageContainer.AddModule(m_probeLibraryMainDlg);
m_iuserManageContainer.AddModule(m_defaultRecipeCtrl);
m_iuserManageContainer.AddModule(m_userManagementDlg);
m_iuserManageContainer.AddModule(m_rawDataExplorer);
m_iuserManageContainer.AddModule(this);
m_iuserManageContainer.OnUpdateUserChange();

void AddModule(IUserManagement user){m_lsUser.Add(user);}
然后调用每个控件自己的OnUpdateUserChange函数
public void OnUpdateUserChange()
{
```

```
    foreach (var item in m_lsUser)
    {
        item.OnUpdateUserChange(USERLEVEL);
    }
}
```

```
 C#中，使用ServiceController类控制windows服务，使用之前要先添加引用：System.ServiceProcess，
 然后在命名空间中引用：using System.ServiceProcess。
private void StartService()
{
    this._controller = new ServiceController("ServicesName");
    this._controller.Start();
    this._controller.WaitForStatus(ServiceControllerStatus.Running);
    this._controller.Close();
}
```

```
C#委托和事件(Delegate、Event、EventHandler、EventArgs)
https://www.cnblogs.com/Scl891004X/p/6142917.html
```

事件

```
class Events : IDrawingObject
{
    event EventHandler PreDrawEvent;

    event EventHandler IDrawingObject.OnDraw
    {
        add
        {
            lock (PreDrawEvent)
            {
                PreDrawEvent += value;
            }
        }
        remove
        {
            lock (PreDrawEvent)
            {
                PreDrawEvent -= value;
            }
        }
    }

}
```

属性

```
value是c#中的“属性”
例如c#某个类中有一个成员变量(字段)，为了安全性，外部如果要访问它，必须通过“属性”来访问：
private int _id;//这是一个成员变量，private表示是私有的，外部不可访问

public int ID
{
     set { _id = value; } //value就是外部为“属性”ID所赋的值

     get { return _id; } //当外部访问“属性”ID时，返回id的值

}

上下文关键字 value 用在普通属性声明的 set 访问器中。 此关键字类似于方法的输入参数
```

```
 DllImport是System.Runtime.InteropServices命名空间下的一个属性类，其功能是提供从非托管DLL导出的函数的必要调用信息。
 DllImport属性应用于方法，要求最少要提供包含入口点的dll的名称。
 DllImport的定义如下：
[AttributeUsage(AttributeTargets.Method)]
     public class DllImportAttribute: System.Attribute
     {
       public DllImportAttribute(string dllName) {…} //定位参数为dllName
       public CallingConvention CallingConvention; //入口点调用约定
       public CharSet CharSet;                                  //入口点采用的字符接
```

```
        public string EntryPoint;                                   //入口点名称
        public bool ExactSpelling;                                 //是否必须与指示的入口点拼写完全一致，默认false
        public bool PreserveSig;                                    //方法的签名是被保留还是被转换
        public bool SetLastError;                                   //FindLastError方法的返回值保存在这里
        public string Value { get {…} }
      }
[DllImport("mpi.dll", EntryPoint = "setMFCmode", CallingConvention = CallingConvention.Cdecl)]     internal static extern void setMFCmode();
[DllImport("mpi.dll", EntryPoint = "ver", CallingConvention = CallingConvention.Cdecl)]            internal static extern StringBuilder ver(int pri
nt);
[DllImport("mpi.dll", EntryPoint = "openDCE", CallingConvention = CallingConvention.Cdecl)]       internal static extern IntPtr openDCE(string pwd
in, string cnfname);
```

```
    class Contact
    {
        public string Name { get; private set; }  //属性 PropertyInfo
        public string Address { get; private set; }//属性
        public string m_sMail;//成员变量 FieldInfo
    }
```

```
public static object Get(string name, Type type)
{
    return mDictionary.GetOrAdd(name, (key) => Activator.CreateInstance(type));
}
```

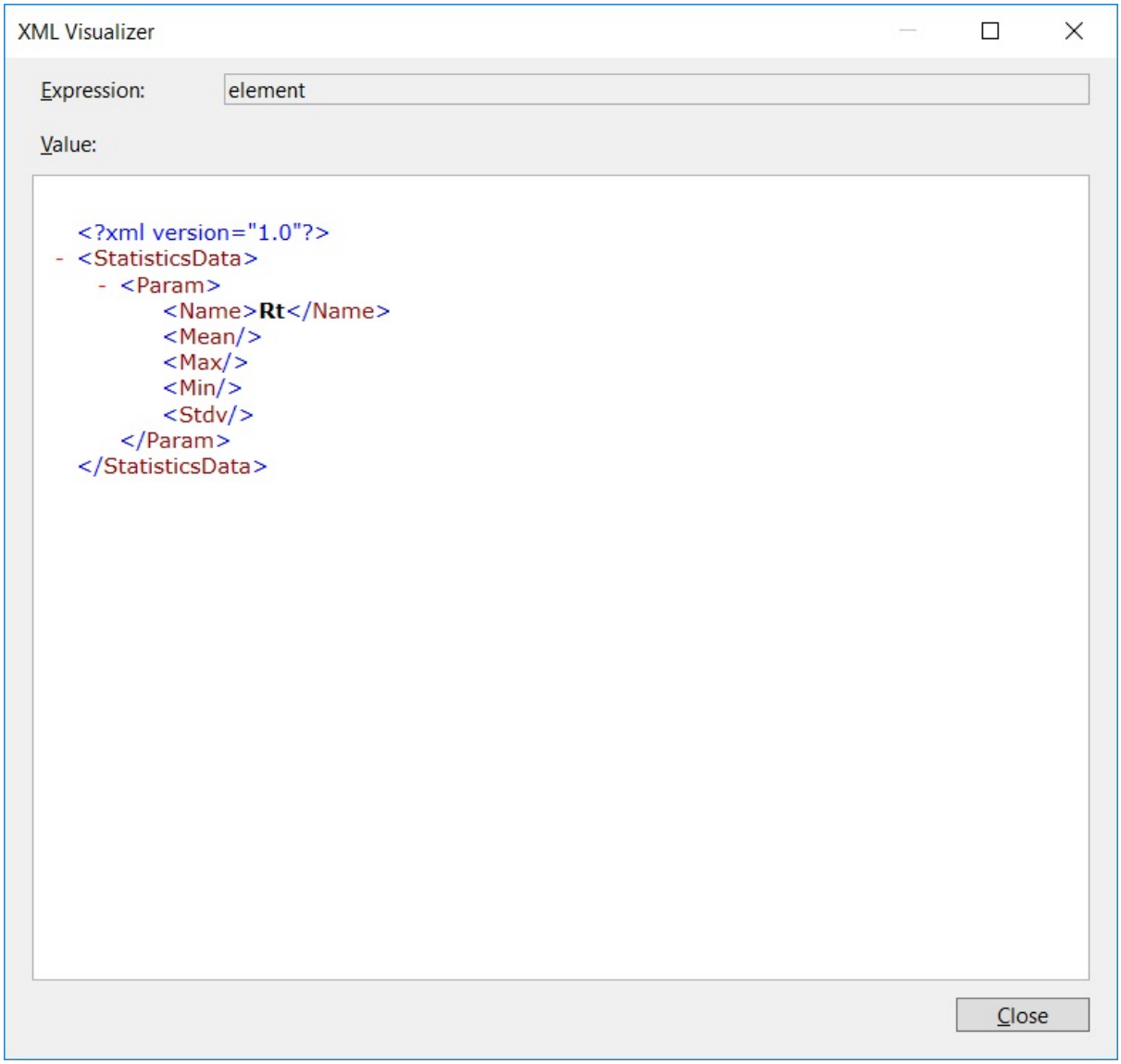
属性接口

```
    public interface IProbeManager
    {
        void UpdateUI();
    }
    public class ProbeManagerFactory
    {
        static public IProbeManager PROBEMANAGER //定义一个属性，其类型是IProbeManager接口
        {
            set;
            get;
        }
    }

    SystemProbeCtrl spc = new SystemProbeCtrl();
    ProbeManagerFactory.PROBEMANAGER = spc; //实现PROBEMANAGER
    快捷键
    1、封装属性的快捷键：ctrl+R+E    :弹出封装好的属性
    2、自动属性快捷键：输入prop+tab+tab
```

XML创建

```
XElement element = new XElement("StatisticsData");//创建XML表格
XElement xe = new XElement("Param",
                                        new XElement("Name", m_lsParamName[nParam]),
                                        new XElement("Mean"),
                                        new XElement("Max"),
                                        new XElement("Min"),
                                        new XElement("Stdv")

                  );

element.Add(xe);//添加子表格到父表格
```

创建一个XML表格，并在这个XML表格中添加另外新的表格



泛型与约束

```
public class ValueWithMinMaxUnit<T> : IXElementSupport where T : struct, IFormattable
    {
        T min;
        T max;
        string unit = "";
        bool minValid = false;
        bool maxValid = false;
        bool unitValid = false;

        public ValueWithMinMaxUnit() { }
        public ValueWithMinMaxUnit(T value, T min, T max)
        {
            Value = value;
            Min = min;
            Max = max;
        }
        public T Value { set; get; }
        public T Min { set{minValid = true; min = value; } get {return min; } }
        public T Max { set { maxValid = true; max = value; } get { return max; } }
        public String Unit
        {
            set
            {
                if (value == null || value == "") return;
                unitValid = true;
                unit = value;
```

```
            }
            get { return unit; }
        }
```

where T : struct, IFormattable //T必须是值类型，且是IFormattable

https://blog.csdn.net/haukwong/article/details/7840158

约束	说明
T：struct	类型参数必须是值类型。可以指定除 Nullable 以外的任何值类型。
T：class	类型参数必须是引用类型，包括任何类、接口、委托或数组类型。
T：new()	类型参数必须具有无参数的公共构造函数。当与其他约束一起使用时，new() 约束必须最后指定。
T：<基类名>	类型参数必须是指定的基类或派生自指定的基类。
T：<接口名称>	类型参数必须是指定的接口或实现指定的接口。可以指定多个接口约束。约束接口也可以是泛型的。
T：U	为 T 提供的类型参数必须是为 U 提供的参数或派生自为 U 提供的参数。这称为裸类型约束.

数据字典

```
public static void AddOrReplace<TKey, TValue>(this IDictionary<TKey, TValue> dico, TKey key, TValue value)
        {
            if(dico.ContainsKey(key))
                dico[key] = value;
            else
                dico.Add(key, value);
        }
```

public interface IDictionary : ICollection>, IEnumerable>, IEnumerable

C# 参数带this是什么意思（扩展方法）

```
using System.Linq;
using System.Text;
using System;

namespace CustomExtensions
{
    //Extension methods must be defined in a static class
    public static class StringExtension
    {
        // This is the extension method.
        // The first parameter takes the "this" modifier
        // and specifies the type for which the method is defined.
        public static int WordCount(this String str)
        {
            return str.Split(new char[] {' ', '.','?'}, StringSplitOptions.RemoveEmptyEntries).Length;
        }
    }
}
namespace Extension_Methods_Simple
{
    //Import the extension method namespace.
    using CustomExtensions;
    class Program
    {
        static void Main(string[] args)
        {
            string s = "The quick brown fox jumped over the lazy dog.";
            //  Call the method as if it were an
            //  instance method on the type. Note that the first
            //  parameter is not specified by the calling code.
            int i = s.WordCount();
            System.Console.WriteLine("Word count of s is {0}", i);
        }
    }
}
```

C# dobule to Decimal

```
decimal dec = 123.45M;
```

WinForm

数字输入框控件(numericUpDown)

```
numericUpDown1.Minimum = 1;  //设置最小值1
numericUpDown1.Maximum = 20;  //设置最大值20
numericUpDown1.DecimalPlaces = 2; //设置空间的DecimaPlaces属性,使空间的数值小数点后两位显示
numericUpDown1.Increment = 1; //设置递增或递减
numericUpDown1.InterceptArrowKeys = true;//设置用户可以通过向上,向下键选择值
```

委托事件回调函数

Lamda表达式

```
Lamda 表达式
To create a lambda expression, you specify input parameters (if any) on the left side of the lambda operator =>,
and you put the expression or statement block on the other side. For example, the lambda expression x => x * x
specifies a parameter that's named x and returns the value of x squared. You can assign this expression to a
delegate type, as the following example shows:
delegate int del(int i);
static void Main(string[] args)
{
    del myDelegate = x => x * x;
    int j = myDelegate(5); //j = 25
}
public bool FloatTopLayer
{
    get => ForwardParameters.GetParameterValue<bool>(CalculationParameters.Names.JxRA_FLOAT_TOP_LAYER, true);
    set => ForwardParameters.SetParameterValue<bool>(CalculationParameters.Names.JxRA_FLOAT_TOP_LAYER, value);
}
```

```
 public class ProbeManagerFactory
    {
        static public IProbeManager PROBEMANAGER
        {
            set;
            get;
        }
    }
```

第二节 反射

Static private 的属性可以通过反射捕捉到。

谈起反射，首先想到的是网上有位大神说过的例子：两个现实中的例子

1、B超：大家体检的时候大概都做过B超吧，B超可以透过肚皮探测到你内脏的生理情况。这是如何做到的呢？B超是B型超声波，它可以透过肚皮通过向你体内发射B型超声波，当超声波遇到内脏壁的时候就会产生一定的“回音”反射，然后把“回音”进行处理就可以显示出内脏的情况了（我不是医生也不是声学专家，不知说得是否准确^_^）。
2、地球内部结构：地球的内部结构大体可以分为三层：地壳、地幔和地核。地壳是固体，地核是液体，地幔则是半液半固的结构（中学地理的内容，大家还记得吧？）。如何在地球表面不用深入地球内部就知道其内部的构造呢？对，向地球发射“地震波”，“地震波”分两种一种是“横波”，另一种是“纵波”。“横波”只能穿透固体，而“纵波”既可穿透固体又可以穿透液体。通过在地面对纵波和横波的反回情况，我们就可以大体断定地球内部的构造了。

大家注意到这两个例子的共同特点，就是从一个对象的外部去了解对象内部的构造，而且都是利用了波的反射功能。在.NET中的反射也可以实现从对象的外部来了解对象（或程序集）内部结构的功能，哪怕你不知道这个对象（或程序集）是个什么东西，另外.NET中的反射还可以运态创建出对象并执行它其中的方法。

反射是.NET中的重要机制，通过反射，可以在运行时获得程序或程序集中每一个类型（包括类、结构、委托、接口和枚举等）的成员和成员的信息。有了反射，即可对每一个类型了如指掌。另外我还可以直接创建对象，即使这个对象的类型在编译时还不知道。

反射的作用（网上广泛流传的）：

作用：1.可以使用反射动态地创建类型的实例，将类型绑定到现有对象，或从现有对象中获取类型；
2.应用程序需要在运行时从某个特定的程序集中载入一个特定的类型，以便实现某个任务时可以使用到反射；
3.反射主要应用与类库，这些类需要知道一个类型的定义，以便提供更多的功能；

什么，不好理解？好吧，我根本背不起来！其实总计起来就是：

作用：反射可以获得程序或程序集的信息，也可以动态的加载程序集，并创建其中某个类型的实例，执行实例中的方法。

接下来我们实例来简单体会一下具体玩法。

反射的用途：
（1）使用Assembly定义和加载程序集，加载在程序集清单中列出模块，以及从此程序集中查找类型并创建该类型的实例。
（2）使用Module了解包含模块的程序集以及模块中的类等，还可以获取在模块上定义的所有全局方法或其他特定的非全局方法。
（3）使用ConstructorInfo了解构造函数的名称、参数、访问修饰符（如pulic 或private）和实现详细信息（如abstract或virtual）等。
（4）使用MethodInfo了解方法的名称、返回类型、参数、访问修饰符（如pulic 或private）和实现详细信息（如abstract或virtual）等。
（5）使用FiedInfo了解字段的名称、访问修饰符（如public或private）和实现详细信息（如static）等，并获取或设置字段值。
（6）使用EventInfo了解事件的名称、事件处理程序数据类型、自定义属性、声明类型和反射类型等，添加或移除事件处理程序。
（7）使用PropertyInfo了解属性的名称、数据类型、声明类型、反射类型和只读或可写状态等，获取或设置属性值。
（8）使用ParameterInfo了解参数的名称、数据类型、是输入参数还是输出参数，以及参数在方法签名中的位置等。

反射用到的命名空间：System.Reflection
System.Type
System.Reflection.Assembly

反射用到的主要类：
System.Type 类——通过这个类可以访问任何给定数据类型的信息。
System.Reflection.Assembly类——它可以用于访问给定程序集的信息，或者把这个程序集加载到程序中。

System.Type类：
System.Type 类对于反射起着核心的作用。但它是一个抽象的基类，Type有与每种数据类型对应的派生类，我们使用这个派生类的对象的方法、字段、属性来查找有关该类型的所有信息。
获取给定类型的Type引用有3种常用方式：
●使用 **C# typeof** 运算符。
Type t = typeof(string);
●使用对象**GetType()**方法。
string s = "grayworm";
Type t = s.GetType();
●还可以调用**Type**类的静态方法**GetType()**。
Type t = Type.GetType("System.String");
上面这三类代码都是获取string类型的Type，在取出string类型的Type引用t后，我们就可以通过t来探测string类型的结构了。
string n = "grayworm";
Type t = n.GetType();

foreach (MemberInfo mi in t.GetMembers())
{
Console.WriteLine("{0}/t{1}",mi.MemberType,mi.Name);
}

Type类的属性：Name 数据类型名
FullName 数据类型的完全限定名(包括命名空间名)
Namespace 定义数据类型的命名空间名
IsAbstract 指示该类型是否是抽象类型
IsArray 指示该类型是否是数组
IsClass 指示该类型是否是类
IsEnum 指示该类型是否是枚举
IsInterface 指示该类型是否是接口
IsPublic 指示该类型是否是公有的
IsSealed 指示该类型是否是密封类
IsValueType 指示该类型是否是值类型
Type类的方法：GetConstructor(), GetConstructors()：返回ConstructorInfo类型，用于取得该类的构造函数的信息
GetEvent(), GetEvents()：返回EventInfo类型，用于取得该类的事件的信息
GetField(), GetFields()：返回FieldInfo类型，用于取得该类的字段（成员变量）的信息
GetInterface(), GetInterfaces()：返回InterfaceInfo类型，用于取得该类实现的接口的信息
GetMember(), GetMembers()：返回MemberInfo类型，用于取得该类的所有成员的信息
GetMethod(), GetMethods()：返回MethodInfo类型，用于取得该类的方法的信息
GetProperty(), GetProperties()：返回PropertyInfo类型，用于取得该类的属性的信息
可以调用这些成员，其方式是调用Type的InvokeMember()方法，或者调用MethodInfo, PropertyInfo和其他类的Invoke()方法。

using System.Reflection;可以加在Assemblyinfo.cs中？

反射在**System.Type**类的应用：

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Reflection;

namespace ReflectionTest
{
    class Contact
    {
        public string Name { get; private set; }
        public string Address { get; set; }

        public string m_sMail;
        private static  string m_sPhone =  "0739" ;
        private static List<String> ticketList = new List<String>()
        {
            "T1","T2","T3","T4","T5","T6","T7","T8","T9","T10",
            "T11","T12","T13","T14","T15","T16","T17","T18","T19","T20",
            "T21","T22","T23","T24","T25","T26","T27","T28","T29","T30",
            "T31","T32","T33","T34","T35","T36","T37","T38","T39","T40",
            "T41","T42","T43","T44","T45","T46","T47","T48","T49","T50"
        };
        public Contact()
        {
        }

        public Contact(string name, string addr)
        {
            Name = name;
            Address = addr;
        }

        public void Print()
        {
            Console.WriteLine("Name: " + Name + "\nAddress: " + Address);
        }
    }
    class Program
    {
        static void Main(string[] args)
        {
            Contact pt = new Contact();
            Type t = pt.GetType();
```

```
ConstructorInfo[] contInfo = t.GetConstructors();
Console.WriteLine("1.Find the constructor in class:");
Console.WriteLine("Number of constructors:" + contInfo.Length);
int num = 0;
foreach(ConstructorInfo c in contInfo)
{
    ParameterInfo[] ps = c.GetParameters();
    Console.WriteLine("This is " + num++ + " constructor:");
    foreach (ParameterInfo pi in ps)
    {
        Console.Write(pi.ParameterType.ToString()+" "+pi.Name + ", ");
    }
    Console.WriteLine();
}

FieldInfo fieldInfo2 = t.GetField("m_sPhone", BindingFlags.NonPublic | BindingFlags.Static);
object value2 = fieldInfo2.GetValue(null);
Console.WriteLine("\nPrivate parameter: " + value2.ToString());

FieldInfo fieldInfo = t.GetField("ticketList", BindingFlags.NonPublic | BindingFlags.Static);
object value = fieldInfo.GetValue(null);
Console.WriteLine("\nPrivate parameter: "+value.ToString());
foreach (String a in (value as List<String>))
{
    Console.Write(a + " ");
}
Console.WriteLine();

FieldInfo[] fldInfo = t.GetFields();
Console.WriteLine();
Console.WriteLine("2.FieldInfo in class:");
foreach (FieldInfo c in fldInfo)
{
     Console.WriteLine(c.ToString());
}

MemberInfo[] membinfo = t.GetMembers();
Console.WriteLine();
Console.WriteLine("3.MemberInfo in class:");
foreach (MemberInfo c in membinfo)
{
    Console.WriteLine(c.ToString());
}

MethodInfo[] methinfo = t.GetMethods();
Console.WriteLine();
Console.WriteLine("4.MethodInfo in class:");
foreach (MethodInfo c in methinfo)
{
    Console.WriteLine(c.ToString());
}

PropertyInfo[] propinfo = t.GetProperties();
Console.WriteLine();
Console.WriteLine("5.PropertyInfo in class:");
foreach (PropertyInfo c in propinfo)
{
    Console.WriteLine(c.ToString());
}

Console.WriteLine("\n6. Using constructor create instance dynamic:");
Type[] tt = new Type[2];
tt[0] = typeof(string);
tt[1] = typeof(string);

ConstructorInfo ctInfo = t.GetConstructor(tt);
object[] obj = new object[2] { "Simon Li", "Shanghai Pudong" };
object o = ctInfo.Invoke(obj);
((Contact)o).Print();

Console.WriteLine("\n7.Create instance with Activator:");
Object[] objs = new object[2] { "Alpha Li", "ShenZheng" };
object ob = Activator.CreateInstance(t, objs);
((Contact)ob).Print();

Console.WriteLine("\n8:Using reflection get object, call properties, methods and fields:");
object obNew = Activator.CreateInstance(t);
FieldInfo fi7 = t.GetField("m_sMail");
fi7.SetValue(obNew, "lipp09");
PropertyInfo pi7 = t.GetProperty("Name");
pi7.SetValue(obNew, "Simon Li");
PropertyInfo pAddr7 = t.GetProperty("Address");
```

```
            pAddr7.SetValue(obNew, "Shanghai",null);
            MethodInfo mi7 = t.GetMethod("Print");
            mi7.Invoke(obNew, null);

            Console.WriteLine("\nI. Using reflection with assembly:");
            Assembly ass = Assembly.Load("ReflectionTest");
            Type tRef = ass.GetType("ReflectionTest.Contact");
            object oRef = Activator.CreateInstance(tRef, "Simon Li", "ShenZheng");
            MethodInfo mi = tRef.GetMethod("Print");
            mi.Invoke(oRef, null);

            Console.WriteLine("\nII.Get all types with full name of dll:");
            Assembly assembly = Assembly.Load("mscorlib.dll");
            Type[] aa = assembly.GetTypes();
            foreach(Type a in aa)
            {
                if(a.FullName == "System.String")
                {
                    Console.WriteLine(assembly.GetName());
                    Console.WriteLine("Get System.String type with reflection");
                }
            }
            Console.Read();
        }
    }
}
```

数据库--从数据库生成.dbml文件以及.cs文件

LINQ之路 9：LINQ to SQL 和 Entity Framework（上）

SqlMetal是跟随VS发布的一个自动工具，可以用来生成数据库的Linq代码。

```
partial class contour
    {
        partial void OnCreated()
        {
            contour_type = 0;
            interval_value = 1.0;
            edge_exclusion = 0;
            flat_exclusion = 0;
            rotation = 0;
            tilt = 0;
            num_intervals = 15;
            display_flag = 0;
            interval_type = 0;
            @override = 1;
            autoscale = 1;

        }

        static public contour Clone(contour from)
        {
            contour ct = new contour()
            {
                contour_type = from.contour_type,
                interval_value = from.interval_value,
                edge_exclusion = from.edge_exclusion,
                flat_exclusion = from.flat_exclusion,
                rotation = from.rotation,
                tilt = from.tilt,
                num_intervals = from.num_intervals,
                display_flag = from.display_flag,
                interval_type = from.interval_type,
                @override = from.@override,
                autoscale = from.autoscale
            };

            return ct;
        }
    }
```

数据库怎么设计的class.cs中的成员变量必须定义在AllDataTable.designer.cs里面。AllDataTable.designer.cs是怎么自动生成的？

```
[global::System.Data.Linq.Mapping.TableAttribute(Name="dbo.contour")]
public partial class contour : INotifyPropertyChanging, INotifyPropertyChanged
{
        private static PropertyChangingEventArgs emptyChangingEventArgs = new PropertyChangingEventArgs(String.Empty);

        private int _contour_sid;

        private System.Nullable<short> _contour_type;

        private System.Nullable<double> _interval_value;

        private System.Nullable<float> _edge_exclusion;

        private System.Nullable<float> _flat_exclusion;

        private System.Nullable<short> _rotation;

        private System.Nullable<short> _tilt;

        private short _num_intervals;

        private int _display_flag;

        private short _interval_type;

        private short _override;

        private short _autoscale;

        private EntitySet<md_collection_summary> _md_collection_summaries;
```

```
        private EntitySet<recipe_desc> _recipe_descs;
```

如何：通过修改 DBML 文件生成自定义代码

你可以从数据库标记语言 (.dbml) 元数据文件生成 Visual Basic 或 C# 源代码。此方法提供了一个在生成应用程序映射代码前自定义默认 .dbml 文件的机会。这是一项高级功能。

此过程中的步骤如下：

1. 生成 .dbml 文件。
2. 使用编辑器修改此 .dbml 文件。请注意，此 .dbml 文件必须通过LINQ to SQL.dbml 文件的架构定义 (.xsd) 文件的验证。有关详细信息，请参阅LINQ to SQL 中的代码生成。
3. 生成 Visual Basic 或 C# 源代码。

下面的示例使用 SQLMetal 命令行工具。有关详细信息，请参阅SqlMetal.exe（代码生成工具）。

示例

下面的代码从 Northwind 示例数据库生成 .dbml 文件。您可以使用数据库的名称或 .mdf 文件的名称作为数据库元数据的源。

复制

```
sqlmetal /server:myserver /database:northwind /dbml:mymeta.dbml
sqlmetal /dbml:mymeta.dbml mydbfile.mdf
```

示例

下面的代码从.dbml 文件生成 Visual Basic 或 C# 源代码文件。

复制

```
sqlmetal /namespace:nwind /code:nwind.vb /language:vb DBMLFile.dbml
sqlmetal /namespace:nwind /code:nwind.cs /language:csharp DBMLFile.dbml
```

```
PILI-E7480\MIURA
sqlmetal /server:PILI-E7480\MIURA /database:emu /dbml:emu.dbml //从数据库生成dbml文件
sqlmetal /namespace:emu /code:emu.cs /language:csharp emu.dbml //从dbml生成.cs文件
```

回调函数

1.回调函数的实现一般在exe程序中[1]。

2.回调函数的调用一般在dll文件中。

3.dll要调用exe中的回调函数，exe必须把回调函数的地址传递给dll。

4.dll有了exe中回调函数的原型和地址，就可以随时调用(触发)该回调函数了。

5.有了1到4，不同编程语言的回调机制，万变不离其宗，一通百通？

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace CallBackFunction
{
    class Program
    {
        static void Main(string[] args)
        {
            Program prog = new Program();   // 在静态函数Main中调用非静态方法时，
                                            // 必须先实例化该类对象,方可调用Add方法
            DllClass dll = new DllClass();

            dll.SetAddCallBack(prog.Add);   // 设置回调
            dll.CallAdd();                  // 触发回调
            Console.Read();                 // 暂停程序
        }

        // 回调函数
        private int Add(int a, int b)
        {
            int c = a + b;
            Console.WriteLine("Add被调用了，a+b={0}", c);
            return c;
        }
    }


    class DllClass  // 假设此类封装在dll中
    {
        // 声明回调函数原型，即函数委托了
        public delegate int Add(int num1, int num2);

        public Add OnAdd = null;   // 此处相当于定义函数指针了

        // 设置回调函数地址
        public void SetAddCallBack(Add add)
        {
            this.OnAdd = add;
        }

        // 调用回调函数
        public void CallAdd()
        {
            if (OnAdd != null)
            {
                OnAdd(1, 99);
            }
        }
    }
}
```

[1]. https://blog.csdn.net/friendan/article/details/42586307 ↩

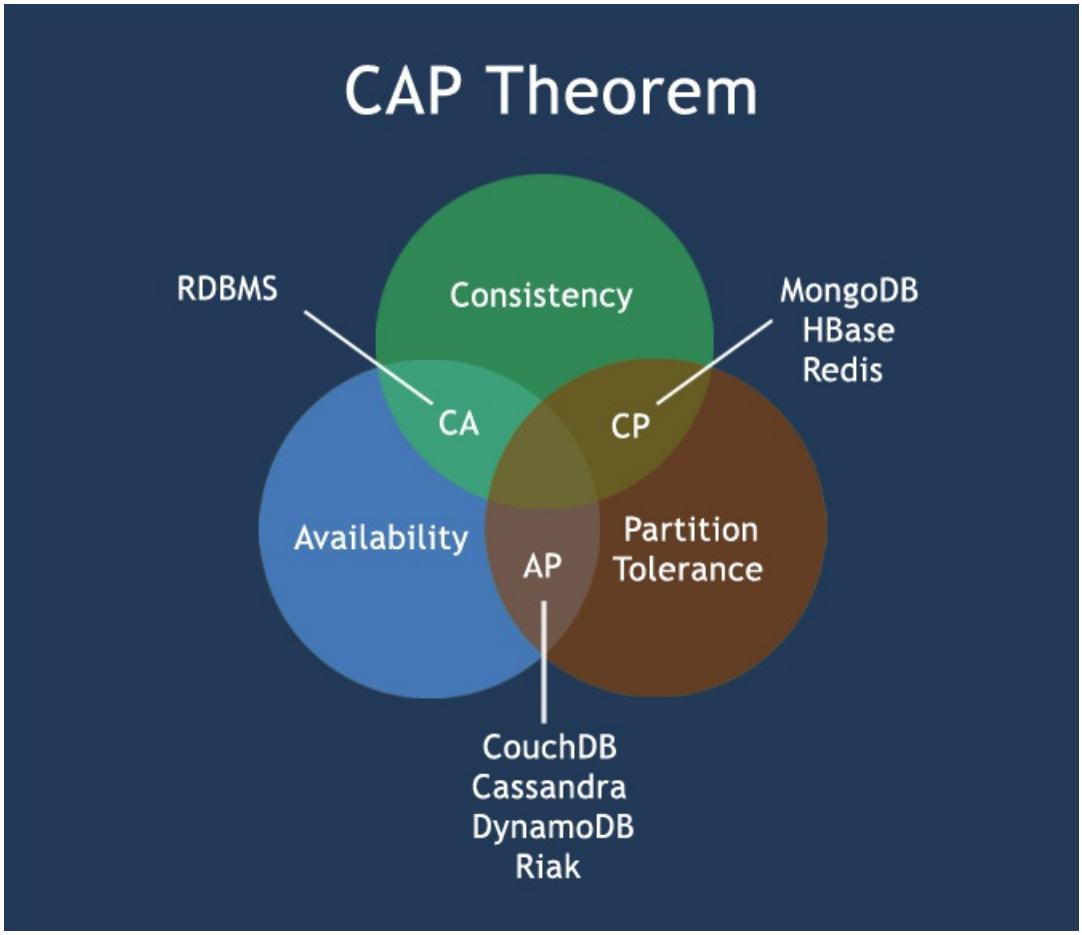
第八章 机器学习

第一节 CNN

第九章 计算机理论

CAP

CAP定理（CAP theorem）
在计算机科学中, CAP定理（CAP theorem）, 又被称作 布鲁尔定理（Brewer's theorem）, 它指出对于一个分布式计算系统来说, 不可能同时满足以下三点:
一致性(Consistency) (所有节点在同一时间具有相同的数据)
可用性(Availability) (保证每个请求不管成功或者失败都有响应)
分隔容忍(Partition tolerance) (系统中任意信息的丢失或失败不会影响系统的继续运作)
CAP理论的核心是：一个分布式系统不可能同时很好的满足一致性，可用性和分区容错性这三个需求，最多只能同时较好的满足两个。
因此，根据 CAP 原理将 NoSQL 数据库分成了满足 CA 原则、满足 CP 原则和满足 AP 原则三 大类：
CA - 单点集群，满足一致性，可用性的系统，通常在可扩展性上不太强大。
CP - 满足一致性，分区容忍性的系统，通常性能不是特别高。
AP - 满足可用性，分区容忍性的系统，通常可能对一致性要求低一些。



NoSQL的优点/缺点

优点:

-
  - 高可扩展性
-
  - 分布式计算
-
  - 低成本
-
  - 架构的灵活性，半结构化数据
-
  - 没有复杂的关系

缺点:

-
  - 没有标准化
-
  - 有限的查询功能(到目前为止)
-
  - 最终一致是不直观的程序

Socket

进程之间的通讯

1.本地进程之间的通讯

本地的进程间通讯(IPC)有很多种方式，但是总结起来为下面四类：

1. 消息传递(管道，FIFO，消息队列)
2. 同步(互斥量，条件变量，读写锁，文件和写记录锁，信号量)
3. 共享内存(匿名的和具名的)
4. 远程过程调用(Solaris门和Sun RPC)

2.网络中进程之间的通讯

网络中进程间如何通讯？首先要解决的问题是如何唯一标识一个进程，否则通讯无从谈起！在本地可以通过进程PID来唯一标识一个进程，但是在网络中这是行不通的。其实TCP/IP协议族已经帮我们解决了这个问题，网络层的“IP地址”可以唯一标识网络中的逐句，而传输层的“协议+端口”可以唯一标识主机中的应用程序(进程)。这样就可以利用三元组(ip地址，协议，端口)来表示网络的进程了，网络中的进行通信就可以利用这个表示与其它进程进行交互。

使用TCP/IP协议的应用程序通常采用应用编程接口：UNIX BSD的套接字(socket)和UNIX System V的TLI(已经被淘汰)，来实现网络进程之间的通信。就目前而言，几乎所有的应用程序都是采用socket，而现在又是网络时代，网络中进程通信是无处不在，这就是我为什么说“一切皆socket”。

2.什么是Socket?

我们知道网络中的进程是通过Socket来通信的，那什么是Socket呢？Socket起源于Unix,而Unix/Linux基本哲学之一就是“一切皆文件”，都可以用“打开open-->读写read/write-->关闭close"模式来操作。我的理解就是Socket就是该模式的一个实现，socket即是一种特殊的文件，一些socket函数就是对其进行的操作(read/write, open, close),这些函数我们在后面进行介绍。

```
socket一词的起源

在组网领域的首次使用是在1970年2月12日发布的文献IETF RFC33中发现的，撰写者为Stephen Carr、Steve Crocker和Vint Cerf。根据美国计算机历史博物馆的记载，Croker写道：“命名空间的元素都可称为套接字接口。一个套接字接口构成一个连接的一端，而一个连接可完全由一对套接字接口规定。”计算机历史博物馆补充道：“这比BSD的套接字接口定义早了大约12年。”
```

3.Socket的基本操作(函数)

既然Socket是"open-write/read-close"模式的一种实现，那么socket就提供了这些操作对应的函数接口，下面以TCP为例，介绍几种基本的socket接口函数。

3.1Socket()函数

```
int socket(int domain, int type, int protocol);
```

socket函数对应于普通文件的打开操作。普通文件的打开操作返回一个文件描述字，而**socket()**用于创建一个socket描述符(socket descriptor)，它唯一标识一个socket。这个socket描述字跟文件描述字一样，后续的操作都有用到它，把它作为参数，通过它来进行一些读写操作。

正如可以给fopen的传入不同参数值，以打开不同的文件。创建socket的时候，也可以指定不同的参数创建不同的socket描述符，socket函数的三个参数分别为：

domain：即协议域，又称为协议族(family)。常用的协议族有，AF_INET、AF_INET6、AF_LOCAL(或称AF_UNIX，Unix域socket)、AF_ROUTE等等。协议族决定了socket的地址类型，在通信中必须采用对应的地址，如AF_INET决定了要用ipv4地址(32位的)与端口号(16位的)的组合、AF_UNIX决定了要用一个绝对路径名作为地址。

type：指定socket类型。常用的socket类型有，SOCK_STREAM、SOCK_DGRAM、SOCK_RAW、SOCK_PACKET、SOCK_SEQPACKET等等(socket的类型有哪些？)。

protocol：故名思意，就是指定协议。常用的协议有，IPPROTO_TCP、IPPTOTO_UDP、IPPROTO_SCTP、IPPROTO_TIPC等，它们分别对应TCP传输协议、UDP传输协议、STCP传输协议、TIPC传输协议(这个协议我将会单独开篇讨论！)。

注意：并不是上面的type和protocol可以随意组合的，如SOCK_STREAM不可以跟IPPROTO_UDP组合。当protocol为0时，会自动选择type类型对应的默认协议。

当我们调用socket创建一个socket时，返回的socket描述字它存在于协议族（address family，AF_XXX）空间中，但没有一个具体的地址。如果想要给它赋值一个地址，就必须调用bind()函数，否则就当调用connect()、listen()时系统会自动随机分配一个端口。

3.2Bind()函数

```
int bind(int sockfd, const struct sockaddr *addr, socklen_t addrlen);
```

网络字节序与主机字节序

主机字节序就是我们平常说的大端和小端模式：不同的CPU有不同的字节序类型，这些字节序是指整数在内存中保存的顺序，这个叫做主机序。引用标准的Big-Endian和Little-Endian的定义如下：

a) Little-Endian就是低位字节排放在内存的低地址端，高位字节排放在内存的高地址端。

b) Big-Endian就是高位字节排放在内存的低地址端，低位字节排放在内存的高地址端。

网络字节序：4个字节的32 bit值以下面的次序传输：首先是0～7bit，其次8～15bit，然后16～23bit，最后是24~31bit。这种传输次序称作大端字节序。由于TCP/IP首部中所有的二进制整数在网络中传输时都要求以这种次序，因此它又称作网络字节序。字节序，顾名思义字节的顺序，就是大于一个字节类型的数据在内存中的存放顺序，一个字节的数据没有顺序的问题了。

所以：在将一个地址绑定到socket的时候，请先将主机字节序转换成为网络字节序，而不要假定主机字节序跟网络字节序一样使用的是Big-Endian。由于这个问题曾引发过血案！公司项目代码中由于存在这个问题，导致了很多莫名其妙的问题，所以请谨记对主机字节序不要做任何假定，务必将其转化为网络字节序再赋给socket。

3.3、listen()、connect()函数

如果作为一个服务器，在调用socket()、bind()之后就会调用listen()来监听这个socket，如果客户端这时调用connect()发出连接请求，服务器端就会接收到这个请求。

```
int listen(int sockfd, int backlog);
int connect(int sockfd, const struct sockaddr *addr, socklen_t addrlen);
```

listen函数的第一个参数即为要监听的socket描述字，第二个参数为相应socket可以排队的最大连接个数。socket()函数创建的socket默认是一个主动类型的，listen函数将socket变为被动类型的，等待客户的连接请求。

connect函数的第一个参数即为客户端的socket描述字，第二参数为服务器的socket地址，第三个参数为socket地址的长度。客户端通过调用connect函数来建立与TCP服务器的连接。

3.4、accept()函数

TCP服务器端依次调用socket()、bind()、listen()之后，就会监听指定的socket地址了。TCP客户端依次调用socket()、connect()之后就想TCP服务器发送了一个连接请求。TCP服务器监听到这个请求之后，就会调用accept()函数取接收请求，这样连接就建立好了。之后就可以开始网络I/O操作了，即类同于普通文件的读写I/O操作。

```
int accept(int sockfd, struct sockaddr *addr, socklen_t *addrlen);
```

accept函数的第一个参数为服务器的socket描述字，第二个参数为指向struct sockaddr *的指针，用于返回客户端的协议地址，第三个参数为协议地址的长度。如果accpet成功，那么其返回值是由内核自动生成的一个全新的描述字，代表与返回客户的TCP连接。

注意：accept的第一个参数为服务器的socket描述字，是服务器开始调用socket()函数生成的，称为监听socket描述字；而accept函数返回的是已连接的socket描述字。一个服务器通常通常仅仅只创建一个监听socket描述字，它在该服务器的生命周期内一直存在。内核为每个由服务器进程接受的客户连接创建了一个已连接socket描述字，当服务器完成了对某个客户的服务，相应的已连接socket描述字就被关闭。

3.5、read()、write()等函数

万事具备只欠东风，至此服务器与客户已经建立好连接了。可以调用网络I/O进行读写操作了，即实现了网咯中不同进程之间的通信！网络I/O操作有下面几组：

read()/write()

recv()/send()

readv()/writev()

recvmsg()/sendmsg()

recvfrom()/sendto()

我推荐使用recvmsg()/sendmsg()函数，这两个函数是最通用的I/O函数，实际上可以把上面的其它函数都替换成这两个函数。它们的声明如下：

```
#include <unistd.h>
```

```
ssize_t read(int fd, void *buf, size_t count);
ssize_t write(int fd, const void *buf, size_t count);

#include <sys/types.h>
#include <sys/socket.h>

ssize_t send(int sockfd, const void *buf, size_t len, int flags);
ssize_t recv(int sockfd, void *buf, size_t len, int flags);

ssize_t sendto(int sockfd, const void *buf, size_t len, int flags,
               const struct sockaddr *dest_addr, socklen_t addrlen);
ssize_t recvfrom(int sockfd, void *buf, size_t len, int flags,
                 struct sockaddr *src_addr, socklen_t *addrlen);

ssize_t sendmsg(int sockfd, const struct msghdr *msg, int flags);
ssize_t recvmsg(int sockfd, struct msghdr *msg, int flags);
```

read函数是负责从fd中读取内容.当读成功时，read返回实际所读的字节数，如果返回的值是0表示已经读到文件的结束了，小于0表示出现了错误。如果错误为EINTR说明读是由中断引起的，如果是ECONNREST表示网络连接出了问题。

write函数将buf中的nbytes字节内容写入文件描述符fd.成功时返回写的字节数。失败时返回-1，并设置errno变量。在网络程序中，当我们向套接字文件描述符写时有俩种可能。1)write的返回值大于0，表示写了部分或者是全部的数据。2)返回的值小于0，此时出现了错误。我们要根据错误类型来处理。如果错误为EINTR表示在写的时候出现了中断错误。如果为EPIPE表示网络连接出现了问题(对方已经关闭了连接)。

其它的我就不一一介绍这几对I/O函数了，具体参见man文档或者baidu、Google，下面的例子中将使用到send/recv。

3.6、close()函数

在服务器与客户端建立连接之后，会进行一些读写操作，完成了读写操作就要关闭相应的socket描述字，好比操作完打开的文件要调用fclose关闭打开的文件。

#include <unistd.h>

int close(int fd);

close一个TCP socket的缺省行为时把该socket标记为以关闭，然后立即返回到调用进程。该描述字不能再由调用进程使用，也就是说不能再作为read或write的第一个参数。

注意：close操作只是使相应socket描述字的引用计数-1，只有当引用计数为0的时候，才会触发TCP客户端向服务器发送终止连接请求。

4、socket中TCP的三次握手建立连接详解

我们知道tcp建立连接要进行“三次握手”，即交换三个分组。大致流程如下：

客户端向服务器发送一个SYN J

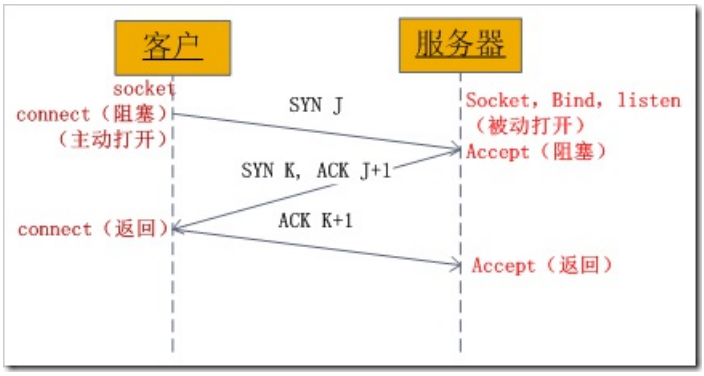
服务器向客户端响应一个SYN K，并对SYN J进行确认ACK J+1

客户端再向服务器发一个确认ACK K+1

这样就完了三次握手，但是这个三次握手发生在socket的那几个函数中呢？请看下图：

如下比喻

```
C:约么？

S:约

C:好的

约会
```



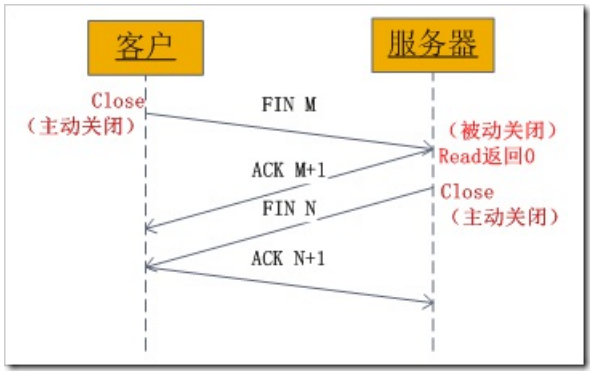
上面介绍了socket中TCP的三次握手建立过程，及其涉及的socket函数。现在我们介绍socket中的四次握手释放连接的过程，请看下图：

从图中可以看出，当客户端调用connect时，触发了连接请求，向服务器发送了SYN J包，这时connect进入阻塞状态；服务器监听到连接请求，即收到SYN J包，调用accept函数接收请求向客户端发送SYN K，ACK J+1，这时accept进入阻塞状态；客户端收到服务器的SYN K，ACK J+1之后，这时connect返回，并对SYN K进行确认；服务器收到ACK K+1时，accept返回，至此三次握手完毕，连接建立。

> 总结：客户端的connect在三次握手的第二个次返回，而服务器端的accept在三次握手的第三次返回。

5、socket中TCP的四次握手释放连接详解

上面介绍了socket中TCP的三次握手建立过程，及其涉及的socket函数。现在我们介绍socket中的四次握手释放连接的过程，请看下图：



图示过程如下：

- 某个应用进程首先调用close主动关闭连接，这时TCP发送一个FIN M；
- 另一端接收到FIN M之后，执行被动关闭，对这个FIN进行确认。它的接收也作为文件结束符传递给应用进程，因为FIN的接收意味着应用进程在相应的连接上再也接收不到额外数据；
- 一段时间之后，接收到文件结束符的应用进程调用close关闭它的socket。这导致它的TCP也发送一个FIN N；
- 接收到这个FIN的源发送端TCP对它进行确认。

这样每个方向上都有一个FIN和ACK。

6.下面给出实现的一个实例

总结：

Client: Connect

Server:accept

Server:send

Client:recv

Client:send

Server:recv

Client:closesocket

```
// Socket-Server.cpp : Defines the entry point for the console application.
//

#include "stdafx.h"
#include <winsock2.h>
#pragma comment(lib,"ws2_32.lib")
//#pragma comment(lib, "WS2_32")
#include "stdio.h"
#include <iostream>
using namespace std;

void main()
{
    WORD myVersionRequenst;
    WSADATA wasData;
    myVersionRequenst = MAKEWORD(1, 1);
    int err;
    err = WSAStartup(myVersionRequenst, &wasData);
    if (!err)
```

```
    {
        printf("open socket!\n");
    }
    else
    {
        printf("Can't open socket!\n");
        return;
    }

    SOCKET serSocket = socket(AF_INET, SOCK_STREAM, IPPROTO_TCP);
    SOCKADDR_IN addr;
    addr.sin_family = AF_INET;
    addr.sin_addr.S_un.S_addr = INADDR_ANY;// htonl(INADDR_ANY);//ip address
    addr.sin_port = htons(6000);//bind port

    if (bind(serSocket, (SOCKADDR*)&addr, sizeof(SOCKADDR)) == SOCKET_ERROR)
    {
        printf("Bind error\n");
        return;
    }
    if (listen(serSocket, 5) == SOCKET_ERROR)
    {
        printf("listen error!\n");
        return;
    }


    //begin listen
    SOCKADDR_IN clientsocket;
    int len = sizeof(SOCKADDR);
    SOCKET serConn = 0;
    while (1)
    {
        //SOCKET serConn = listen(serSocket, 5);
        serConn = accept(serSocket, (struct sockaddr*)&clientsocket, &len);
        if (serConn == INVALID_SOCKET)
        {
            printf("Failed accept()\n");
            return;
        }
        //SOCKET serConn = connect(serSocket, (SOCKADDR*)&clientsocket, &len);
        char sendbuf[100];
        sprintf_s(sendbuf, "welcome %s!", inet_ntoa(clientsocket.sin_addr));
        send(serConn, sendbuf, strlen(sendbuf) + 1, 0);

        char receiveBuf[100];
        recv(serConn, receiveBuf, strlen(receiveBuf) + 1, 0);
        printf("%s\n", receiveBuf);
        closesocket(serConn);//close
        WSACleanup();//release resource
    }
}
```

```
// SocketClient.cpp : Defines the entry point for the console application.
//

#include "stdafx.h"
#include <WinSock2.h>
#include <stdio.h>
#pragma comment(lib, "ws2_32.lib")


void main()
{
    int err;
    WORD versionRequired;
    WSADATA wsaData;
    versionRequired = MAKEWORD(1, 1);
    err = WSAStartup(versionRequired, &wsaData);//protocal version info

    if (!err)
    {
        printf("Client socket opened!\n");
    }
    else
    {
        printf("Client socket can't open!\n");
        return;
    }

    SOCKET clientSocket = socket(AF_INET, SOCK_STREAM, 0);
```

```
    SOCKADDR_IN clientsock_in;
    clientsock_in.sin_addr.S_un.S_addr = inet_addr("127.0.0.1");
    clientsock_in.sin_family = AF_INET;
    clientsock_in.sin_port = htons(6000);
    connect(clientSocket, (SOCKADDR*)&clientsock_in,sizeof(SOCKADDR));//start connect

    char receivebuf[100];
    recv(clientSocket, receivebuf, 101, 0);
    printf("%s\n", receivebuf);

    send(clientSocket, "Hello, this is client\n", strlen("Hello, this is client\n") + 1, 0);
    closesocket(clientSocket);
    WSACleanup();
    return;
}
```

同步与内核对象

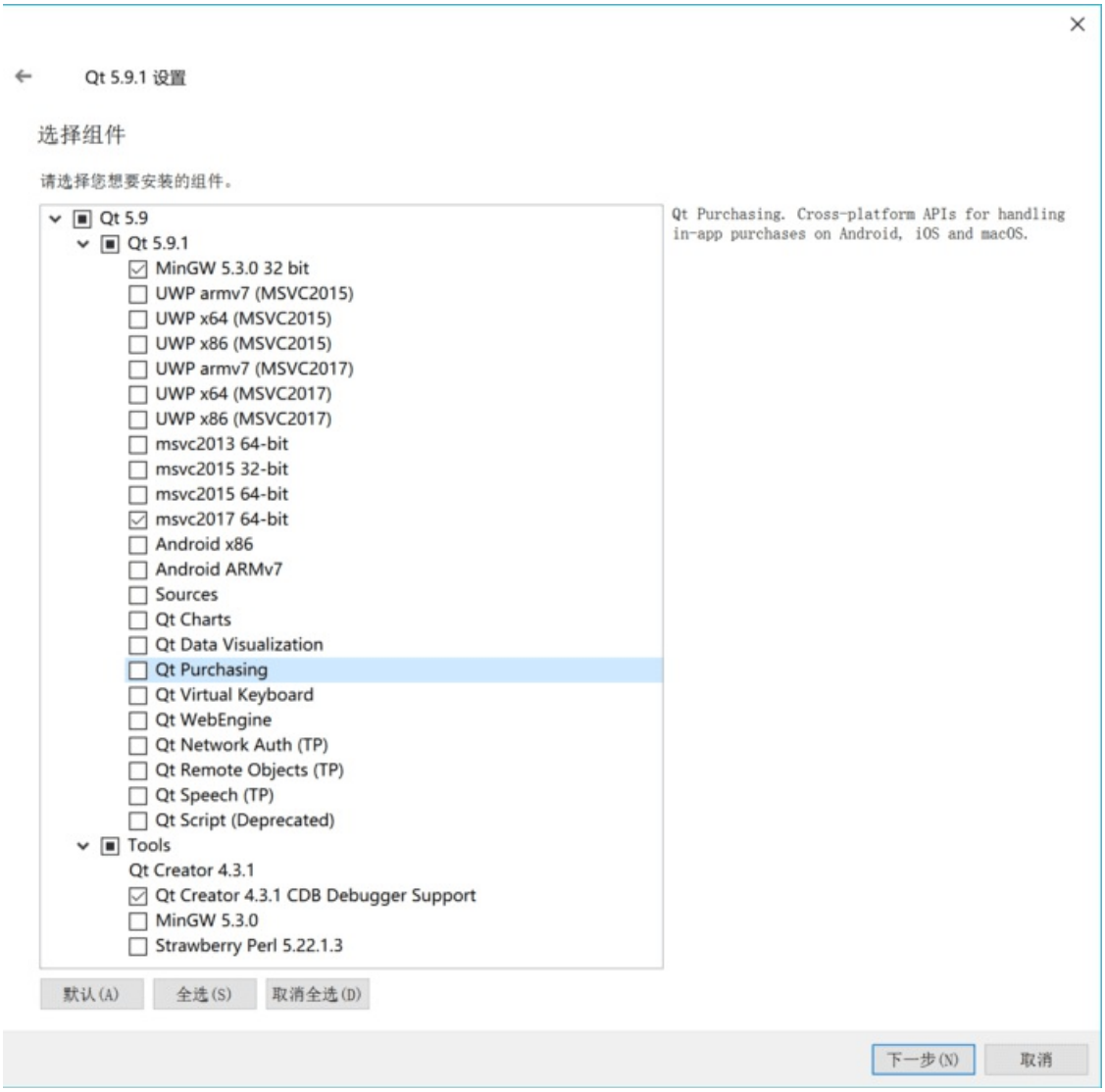
第十章 QT

之所以会涉及Qt这一章，是因为能利用Qt进行跨平台开发，最终还是要转到Linux上进行UI开发，因此现在需要学会使用Qt。

Qt Creator

VS2017上面安装QT

2.1 打开Qt下载页面：https://download.qt.io/
2.2 进入路径：/official_releases/qt/5.9/5.9.1/，下载文件 qt-opensource-windows-x86-5.9.1.exe
然后安装,Qt5.9.1不需要全选，不然需要13G的空间，安装时间也太长，就选下面两项



安装好之后

1. 打开Visual Studio 2017，点击菜单栏中的“工具”，找到“扩展和更新”：
2. 在扩展和更新中，点击左侧目录进入Visual Studio Marketplace，搜索关键词Qt，第一个结果就是Qt Visual Studio Tools，安装：
3. 安装完成后，菜单栏中会出现新的一项：Qt VS Tools，点击进入Qt Options，点击“Add”，在path一栏中设置Qt 5.9.1的安装路径，如果路径不正确，会红字提示你没有找到某个特定的文件。
4. 以上配置工作完成后，新建项目，在Visual C++类别中选择Qt，然后选择Qt GUI Application，按照默认配置，即可生成项目文件。
5. 使用VS开发Qt项目，比起Qt Creator，有一个不太方便的地方就是对于ui文件的编辑。在项目文件目录中找到.ui后缀的文件，双击打开，会发现它会用另一个软件：Qt Designer打开.ui文件，有一点影响开发效率，不过总体来说还算可以接受。
6. 完成自己的项目代码后，我们选择生成解决方案，但是会报错，原因是SDK的配置不合适。在VS菜单栏中找到“项目”，点击“重定解决方案目标”，选择Windows10 的SDK版本，再次编译，成功

第十一章 Python

PyBasic

```
dicFile = open('train_1.txt', 'r')#打开数据
print '开始装载数据...'
txtDict = {}#建立字典
while True:
    line = dicFile.readline()
    if line == '':
        break
    index = line.find('\t')#以tab键为分割
    key = line[:index]
    value = line[index:]
    txtDict[key] = value#加入字典
dicFile.close()
##查找字典
srcFile = open('train1.txt', 'r')#要匹配的key
destFile = open('match.txt', 'w')#符合字典的写入里面
while True:
    line = srcFile.readline()
    if line == '':
        break
    index = line.find(' ')
    key = line[:index]
    if txtDict.has_key(key):
        destFile.write(key)
        destFile.write(txtDict[key])
    else:
        badFile.write(key)
        badFile.write('\n')
print '全部完成！'
destFile.close()
srcFile.close()
```

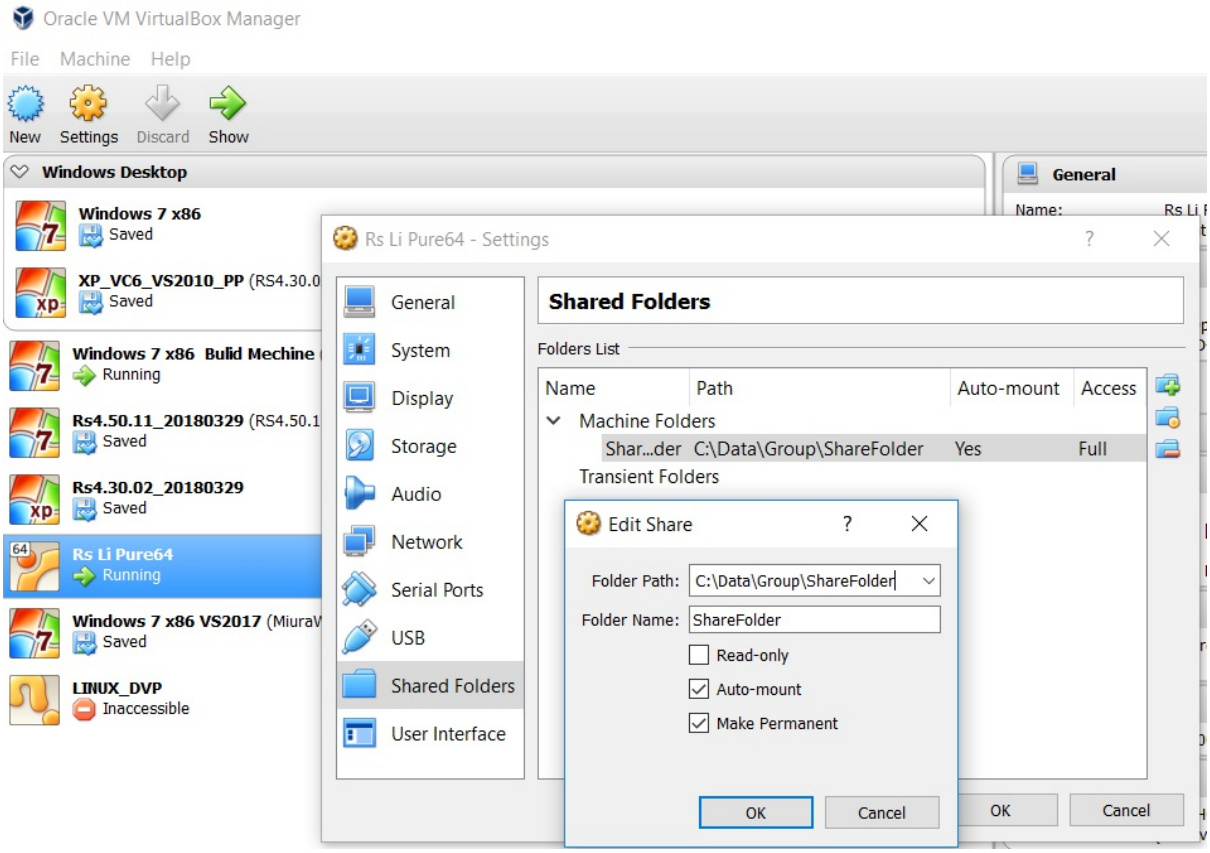
Linux Basic

Ubuntu VirtualBox上建立share folder[1]

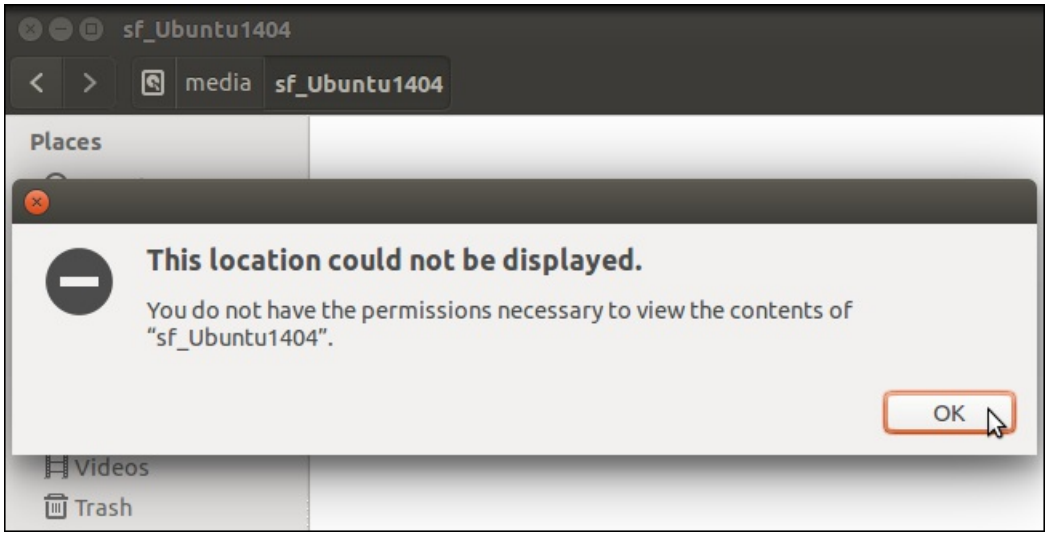
https://www.howtogeek.com/187703/how-to-access-folders-on-your-host-machine-from-an-ubuntu-virtual-machine-in-virtualbox/

1通过VirtualBox setting中的shared folded选定windows中的一个文件作为共享文件夹

要勾Auto-mount, 否则在ubuntu中显示不出来



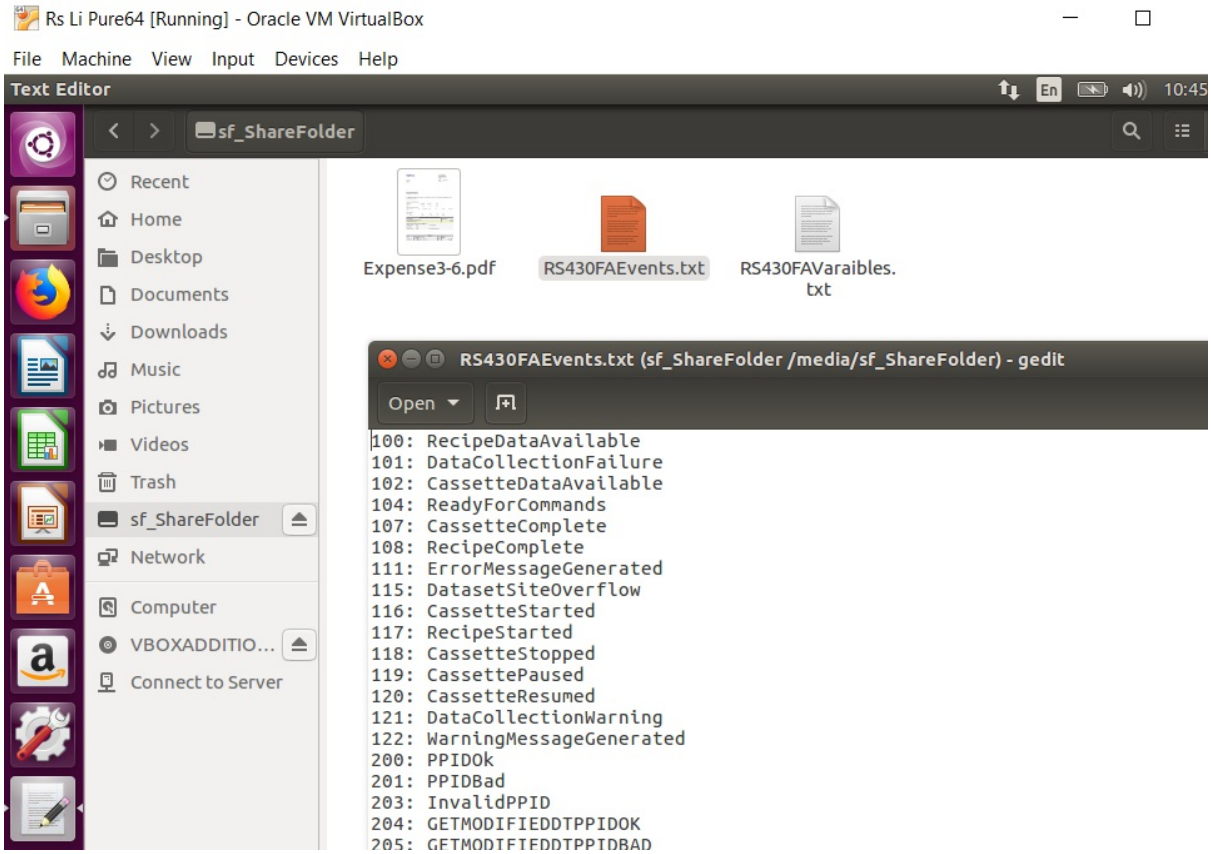
2.打开ubuntu, 会发现medis中会有新的文件夹, sf_ShareFolder,但是不能打开。



3.需要把我们的ubuntu虚拟机加入到vboxsf

命令:sudo adduser lipp vboxsf

4.重启虚拟机就可以用了



Linux下文本排序(文本中数字排序)

sort -n RS430FAVaraibles.txt -o RS430FAVaraiblesSorted.txt

结束