**POCO C++库学习和分析 --  随机数和数字摘要**

           在程序设计时，有时候我们需要生成随机数和数字摘要。在Poco库中，也提供了上述功能，下面我们一一叙述：

**1. 随机数生成**

           Poco中生成随机数的类为Poco::Random类。它根据PRNG([pseudo random number generator](http://en.wikipedia.org/wiki/Pseudorandom_number_generator) )算法设计，采用了一个累加的非线性反馈算法。PRNG算法可以产生0 ~ 2^31之间的随机数整数。  
           在接口上Poco::Random提供了一些函数，可以使使用者直接得到其他形式的随机数。如char, bool, float 和 double 类型。另外Poco库中还提供了RandomInputStream类，用于Poco::Random类的流操作。  
  
  
**成员函数：**  
           1. void seed(Poco::UInt32 seed)  
           根据给定的种子值生成随机数。

           2. void seed()  
           使用任意值(从RandomInputStream类中获取)生成随机数。

           3. 默认的构造时，Poco::Random类采用当前的时间和日期生成随机数。如果想要更好的随机效果，需要显式的调用seed()方法

           4. UInt32 next()  
           返回0 ~ 2^31之间的随机整数

           5. UInt32 next(UInt32 n)  
           返回0 ~ n之间的随机整数

           6. char nextChar()  
           返回随机Char值

           7. bool nextBool()  
           返回随机bool值

           8. float nextFloat()  
           返回随机float值，范围0 ~ 1

           9. double nextDouble()  
           返回随机double值，范围0 ~ 1  
  
  
           下面是关于Random的一个例子：

**[cpp]** [view plaincopy](http://blog.csdn.net/arau_sh/article/details/8698418)

1. #include "Poco/Random.h"
2. #include "Poco/RandomStream.h"
3. #include <iostream>
4. **using** Poco::Random;
5. **using** Poco::RandomInputStream;
6. **int** main(**int** argc, **char**\*\* argv)
7. {
8. Random rnd;
9. rnd.seed();
10. std::cout << "Random integer: " << rnd.next() << std::endl;
11. std::cout << "Random digit: " << rnd.next(10) << std::endl;
12. std::cout << "Random char: " << rnd.nextChar() << std::endl;
13. std::cout << "Random bool: " << rnd.nextBool() << std::endl;
14. std::cout << "Random double: " << rnd.nextDouble() << std::endl;
15. RandomInputStream ri;
16. std::string rs;
17. ri >> rs;
18. **return** 0;
19. }

**2. 密码散列**

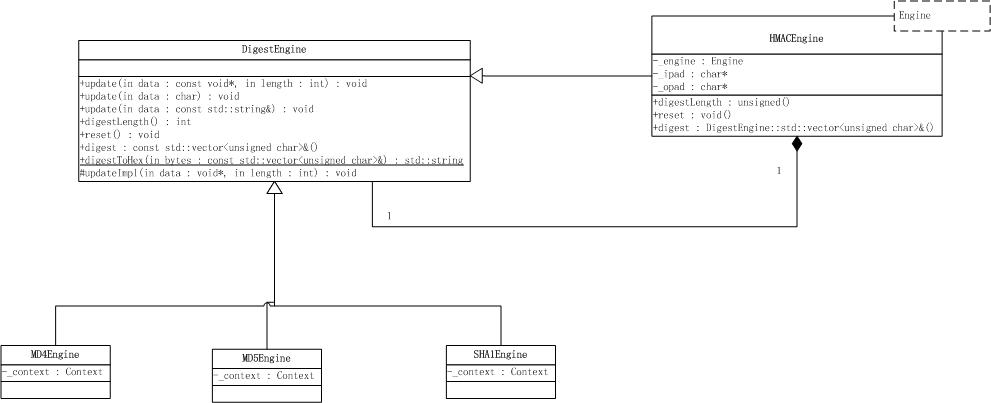
           下面这段是Wiki上关于密码散列的介绍：  
           A cryptographic hash function is a hash function with certain additional security properties to make it suitable for use as a primitive in various information security applications, such as authentication and message integrity. A hash function takes a long string (or message) of any length as input and produces a fixed length string as output, sometimes termed a message digest or a digital fingerprint. Wikipedia

**2.1 概述**

           密码散列（cryptographic hash）是将目标文本转换成具有相同长度的、不可逆的杂凑字符串（或叫做消息摘要）。它有两个特点：  
           1、哈希算法往往被设计成生成具有相同长度的文本  
           2、哈希算法是不可逆的。(因为如果可逆，那么哈希就是世界上最强悍的压缩方式——能将任意大小的文件压缩成固定大小)  
  
           密码散列是一个多对一映射，好的哈希算法应该对于输入的改变极其敏感。Poco中实现了被广泛使用的密码散列函数(cryptographic hash functions), 包括了MD4,[MD5](http://baike.baidu.com/view/7636.htm)  
和[SHA1](http://baike.baidu.com/view/1228622.htm)。另外还提供了HMACEngine类实现了[HMAC](http://baike.baidu.com/view/1136366.htm)功能。HMAC全称为Hash-based Message Authentication Code,HMAC运算利用哈希算法，以一个密钥和一个消息为输入，生成一个消息摘要作为输出。

**2.2 DigestEngine类**

           Poco::DigestEngine类为所有的消息摘要类定义了通用接口。  
           1. unsigned digestLength()  
           用于获取不同消息摘要算法生成消息摘要的长度。  
           2. const Digest& digest()  
           获取消息摘要内容  
           3. update(const void\* data, unsigned length)  
           更新消息摘要内容  
  
  
让我们来看一下DigestEngine类的类图。



下面是Poco中相关类的一些例子：

**[cpp]** [view plaincopy](http://blog.csdn.net/arau_sh/article/details/8698418)

1. #include "Poco/HMACEngine.h"
2. #include "Poco/SHA1Engine.h"
3. **using** Poco::DigestEngine;
4. **using** Poco::HMACEngine;
5. **using** Poco::SHA1Engine;
6. **int** main(**int** argc, **char**\*\* argv)
7. {
8. std::string message1("This is a top-secret message.");
9. std::string message2("Don't tell anyone!");
10. std::string passphrase("s3cr3t"); // HMAC needs a passphrase
11. HMACEngine<SHA1Engine> hmac(passphrase); // we'll compute a HMAC-SHA1
12. hmac.update(message1);
13. hmac.update(message2);
14. **const** DigestEngine::Digest& digest = hmac.digest();
15. // finish HMAC computation and obtain digest
16. std::string digestString(DigestEngine::digestToHex(digest));
17. // convert to a string of hexadecimal numbers
18. **return** 0;
19. }

**2.3 与DigestEngine类相关的流(DigestInputStream/DigestOutputStream)**

           可以通过Poco::DigestInputStream和Poco::DigestOutputStream类对DigestEngine类进行输入输出操作。过程很简单，只要在在构造Stream时，把相关的DigestEngine类传入即可。需要注意的是，在向DigestOutputStream类写入后，要及时调用flush函数，以确保Stream把所有数据都输入进DigestEngine类。  
  
           下面是相关的一个例子：

**[cpp]** [view plaincopy](http://blog.csdn.net/arau_sh/article/details/8698418)

1. #include "Poco/DigestStream.h"
2. #include "Poco/MD5Engine.h"
3. **using** Poco::DigestOutputStream;
4. **using** Poco::DigestEngine;
5. **using** Poco::MD5Engine;
6. **int** main(**int** argc, **char**\*\* argv)
7. {
8. MD5Engine md5;
9. DigestOutputStream ostr(md5);
10. ostr << "This is some text";
11. ostr.flush(); // Ensure everything gets passed to the digest engine
12. **const** DigestEngine::Digest& digest = md5.digest(); // obtain result
13. std::string result = DigestEngine::digestToHex(digest);
14. **return** 0;
15. }

（版权所有，转载时请注明作者和出处  <http://blog.csdn.net/arau_sh/article/details/8698418>）