编程核心概念

计算机技术非常庞大，而且仍在快速更新，面对越来越多的新技术，我们似乎跟不上了。如何面对新技术，如果面对换工作时的技术迁移，是必须要考虑的。
优秀的程序员不会被技术迭代淘汰掉，也不惧怕切入新的技术领域。掌握一些核心的概念，可迁移的能力，是我们从容应对这些的底气。之后要介绍的内容，正是基于此而产生的，我个人认为比较重要的一些核心概念。这些概念虽然简单，但需要我们时常放在心上。

protocol

协议，或者说约定，是我们构建任何系统的基础，我们制定标准，在标准的指导下通力合作，构建出一个个庞杂的系统。比如网络协议 tcp/ip，编码标准，编程语言，甚至我们平时说的普通话，都可以算作是协议。
协议本身是很简单的，都是规定死的东西，重要的是协议为什么是这样设计的，使用了哪些技巧，原理等，所以它的发展历程也很重要。

encoding/decoding

编码与解码无处不在，在我的概念里，编码/解码是协议的实现，比如utf-8的编解码实现了utf-8这个标准。
编解码处理的是数据，而且一般成对出现，是可逆的，比如软件工程里的逆向工程；但有的是不可逆的，比如有损压缩，你需要借助一些插值算法，才能把数据补回去，补回去数据也并非原数据；再比如hash，只有编码过程，没有解码过程。
之前面试的时候，有面试者说SHA1 hash是加密算法，虽然加密算法也是encoding的过程，但区别是，加密必然伴随着解密，即decoding，所以hash只能算编码算法。这里贴一段英文的解释[1]，因为大部分中文教程里，都有 ~~md5是加密算法~~ 这样的错误描述，导致很多人对此深信不疑。

```
By definition, a hash function is not encryption.

> Encryption is the process of encoding messages (or information) in such
a way that eavesdroppers cannot read it, but that authorized parties can.

and

> Hash function is an algorithm that takes an arbitrary block of data and
returns a fixed-size bit string, the cryptographic hash value, such that
any change to the data will (with very high probability) change the hash
value.

Encryption provides confidentiality while hash functions provide
integrity.

Hash functions are used alongside encryption for their integrity
capabilities.
```

我们在阅读或者debug代码的时候，多去寻找编解码的函数，如此，数据的边界，流向，变化过程会比较清晰，便于我们快速定位出错的位置。

举个乱码的例子，很多人都遇到过乱码，有些情况比较简单，有些则看起来复杂。比如，同样的代码，在虚拟机上的输出是乱码，在本地PC上的输出却是正常的。如果我们只关注到代码是一样的，你会觉得这问题出的很是莫名其妙对不对，但如果你注意到，我们从程序里输出到terminal的内容，是会被terminal解码的，这时你应该会想到，是不是本地编码和远程虚拟机设置的编码不一致导致了这种差异。

binary

二进制是我们在计算机人门时最先接触到的概念，虽然我们熟知二进制是信息最基础的表现形式，但我们并不一定时常能以这样的认知去看待事物。
以编程语言里的类型来举例，在我们的认知里，类型通常是不能随意转换的，你会遇到各种编译错误，对不对？但实际上是可以的。所谓类型，只是编译器用来约束变量边界的规范，任何类型，其本质都是二进制，我们可以用 `int32` 来存储1-4个 `char` ，也可以用一个 `double` 来存储1-4个 `int32` , 还可以用 `short` 来存储 `float` 以降低 `float` 的精度，从而达到降低存储空间的目的等等。二进制级别的运算是很有意思的，可以直接让我们操作内存，在性能和存储空间方面做更深入的优化，也可以做一些精妙的hack。但并不是所有语言都提供这种自由。

```
// c++
// 32bit float 降精度，用16bit short 存，节省存储空间
void FPC::__32To16(const float x, unsigned short &res)
{

    //little endian
    assert(GABS(x)  < 1.0f);
    int  nTmp = static_cast<int>(x*32768);
    nTmp += 32768;
    res = nTmp & 65535;

}
```

input/output

i/o 无处不在，但我们不一定会特别注意到它。比如磁盘io，键盘(input)+显示器(output)，网络io等。io一般发生在软件与软件或者软件与硬件的边界处，对我们理解软件 / 系统架构至关重要，自己做架构设计时候也要时常注意io的边界。

emulation

"一切能用硬件实现的东西，都可以用软件来模拟“。这是一位学长教给我的至今印象很深的一句话。就像摩尔定律一样，这句话相当于一种论断，预言，它能带给你信心，也会促使你去寻找反例。我们可以用软件构建出一个和现实世界一模一样的虚拟世界: 仿真系统，虚拟化，自动化系统等等。
除了使用软件模拟现实世界这方面，模拟的概念也时常出现在使用软件模拟软件方面。比如，模拟浏览器访问网站；模拟软件系统中的某个组件，用来测试系统中的其他组件或者伪装身份等等。

data-structure and algorithm

程序设计 = 数据结构 + 算法， 在C语言的第一堂课里，老师应该会给你们讲这个公式，它们的重要性不言而喻。数据结构用来组织数据，堆栈，数组，链表，图，树等等，我们需要掌握很多种数据结构，并关注它们

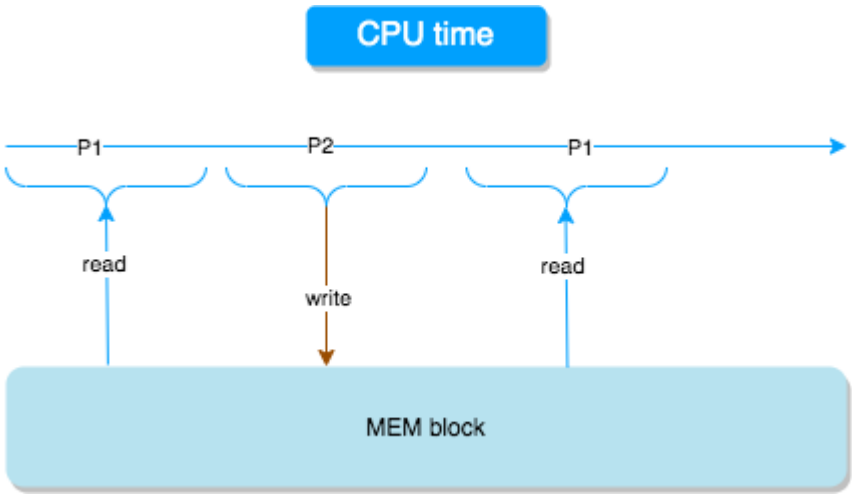
的使用场景。算法和数据结构是相辅相成的，合在一起构成解决问题的方案，也可以认为是解耦的，同一个算法可以使用不同的数据结构来实现。

time-sharing and asynchronous

我们熟知的操作系统都属于分时操作系统，这种分时机制可以让多个进程"同时"跑，cpu资源按时间分片，分配给不同的进程，以达到多任务的效果, 它是并发的基础。而我们熟知的异步, 和并发的概念是不同的，异步是一种编程模型，你可以开一个子进程去执行长操作（比如磁盘io），而不用阻塞长操作之后的逻辑。

lock

对于cpu资源，操作系统会把它按时间分片，然后分配给所有线程，不存在竞争，因为时间是不可能重叠的，但像内存，文件这类资源，是没办法按时间分片的，在并发的时候就有可能产生数据竞争（data race）。



如图所示，P1, P2相当于"同时"在访问这个内存块，因为P1的读还未结束，P2就开始修改它了。因此，我们需要对资源加锁，保证P1在完成读之前，P2是是不能写的。

centralize/decentralize and distribution

我们经常会听到分布式架构, 但分布式（distribution）和去中心化（decentralize，也可以称非集中式）不是一个概念。分布式是指整个系统被分散到不同的位置，用来提升性能或者提高可靠性（比如git），而去中心化是指的是不存在一个单一的管理实体去管理整个系统，去中心化依赖多个相互独立的管理实体共同管理整个系统。SDN是中心化（也称集中式）的，区块链是去中心化的，也是分布式的。

参考资料

[1] is-sha-1-encryption