Algorithmn

438. Find All Anagrams in a String

```
class Solution {
public:
    std::vector<int> findAnagrams(std::string s, std::string p) {
        std::map<char, int> pMap;
        std::vector<int> result;
        if(s.length() < p.length()) return result;
        for(auto& c : p){
            pMap[c]++;
        }

        int i = 0;
        while(i <= s.length() - p.length()){
            auto mapTmp = pMap;
            int j = 0;
            bool finished = true;
            for(j = i; j < i + p.length(); j++){
                auto iter = mapTmp.find(s[j]);
                if(iter == mapTmp.end()){
                    i = j + 1;
                    finished = false;
                    break;
                }else{
                    iter->second--;
                }
            }
            if(!finished) continue;
            bool match = true;
            for(auto &it : mapTmp){
                if(it.second != 0){
                    i++;
                    match = false;
                    break;
                }
            }
            if(match){
                result.push_back(i);
                i++;
            }
```

```
        }
        return result;
    }
};

class Solution2 {
public:
    std::vector<int> findAnagrams(std::string s, std::string p) {
        std::map<char, int> pMap;
        std::map<char, int> sMap;
        std::vector<int> result;
        int sLen = s.length();
        int pLen = p.length();
        if(sLen < pLen) return result;
        for(int i = 0; i < pLen; i++){
            sMap[s[i]]++;
            pMap[p[i]]++;
        }

        int i = 0;
        for(i = pLen; i < sLen; ++i){
            if(sMap == pMap){
                result.push_back(i-pLen);
            }
            sMap[s[i]]++;
            sMap[s[i-pLen]]--;
            if(sMap[s[i-pLen]] <= 0){
                sMap.erase(s[i-pLen]);
            }
        }
        if(sMap == pMap){
            result.push_back(i-pLen);
        }

        return result;
    }
};
```

这个题目是要找出字符串s中所有具有类似字符串p的下标。这里刚开始的解法是用一个map来保存p中字符，然后不断轮询s中的字符串来判断字符是否全在map中，这种解法可以得出正确的结果，但是效率太低，很容易超时。在后续的改进算法中，可以设计两个map，分别表示s中一段字符串代表的字符串，另一个代表字符串s，可以通过判断两个map是否相等来确定是否是异构结构的字符串。

Review

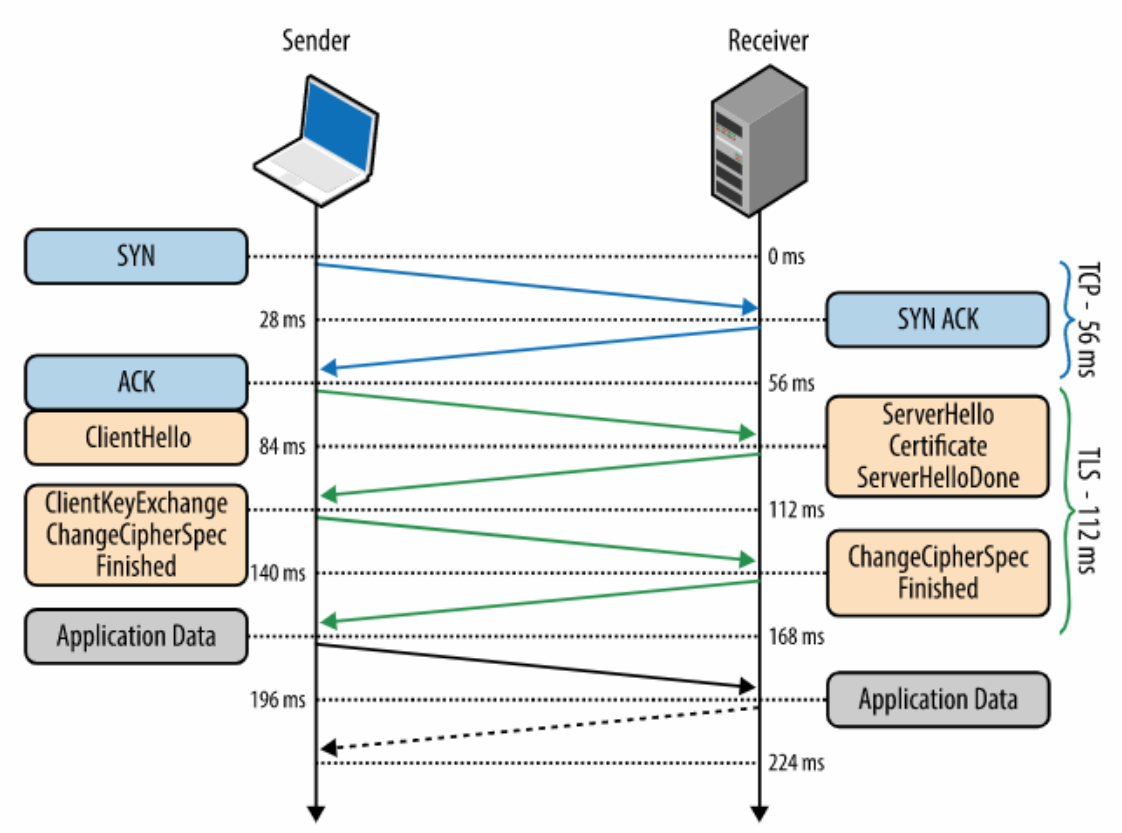
Understanding Browser Networking and Optimizing Requests for HTTP/1.1 and HTTP/2

这是一篇关于浏览器工作流程的入门文章。其介绍了HTTP协议的基础--TCP/IP协议，同时介绍了HTTP协议的并发问题，以及目前如何解决这些问题。

HTTP协议的底层协议是TCP/IP， 其中ip唯一标志互联网上的一台主机，而TCP负责将IP层传递来的报文组成包，保证有序性可靠性和完整性。在TCP/IP通过三次握手建立连接，通告可以接受的窗口大小，通过这个滑动窗口控制发送报文的速率，发送报文的过程中需要接收到对端的ack报文才会接着发送下一个报文。在发送开始时是通过不断试探，不断增大发送报文的大小，遇到网络颠簸，接受ack超时，就会减半报文大小，这是TCP/IP协议中的慢启动及拥塞控制算法。

当在浏览器中输入URL时，第一步是在DNS根据域名查找IP地址，由于每次查找IP地址的成本很高，因此一般在浏览器，主机，域名服务器各个层级之间建立了缓冲，可以通过缓冲查找常见的域名的IP地址，而提高了查询的效率。

建立好TCP/IP连接之后，一般还会再次基础之上进行TLS握手，需要额外两个步骤才能建立安全的连接，其过程如下：



一个浏览器可能同时发起多个Request，如果一个请求就发起一次连接，那么会有很多次的三次握手即TLS建立过程，造成较大的额外开销。因此一般会服用连接，一种极端的做法是在与一个host的通信过程复用一个连接，但是这样会造成请求的串行化，响应变慢。因此一个比较好也是目前使用的方式是建立多个但是有限数量的连接(6个)，请求复用这些连接，那么既可以减少减少创建连接的开销，又可以提高并发度。但在HTTP/2协议中，可以利用多路分发，即只创建一个连接，请求之间不再具有串行性，既可以减少连接数量，也可以提高并发效率。

Tech

事务的隔离级别

串行化操作可以保证事务的一致性，但是其不能并发操作，将会导致性能低下，因此可以在保证弱一致性的前提下提供多个事务隔离级别，提高并发性。但是弱一致性级别下回增加程序的复杂性，给程序员增加额外的负担。事务具有以下隔离级别：

1. 可串行化
2. 可重复读，只允许读取已提交的数据，而且在一个事务两次读取一个数据项期间，其他事务不得更新该数据。
3. 读已提交，只允许读取已提交的数据，但不要求可重复读
4. 读未提交

隔离性级别的实现

并发机制的目的是获取高的并发性，同时保证所产生的调度室冲突可串行化或视图可串行化的，并且是无级联的。并发控制机制如下：

1. 锁：锁可以封锁其访问的数据项，而不用封锁整个数据库。二阶段锁协议分为两个阶段，第一阶段获取锁，第二阶段释放锁。锁有两种类型：共享锁和排他锁。共享锁可以多个锁共享，用于多个事务的读数据项，排他锁只能独占，使用于写数据项，多个共享锁可以实现并发读。
2. 时间戳：
3. 多版本和快照隔离

Share

interface是一种类型

interface是一种类型：`type I interface{}`,interface是一种具有方法的类型， 如果一个interface没有任何方法则是一个empty interface。如果一个类型实现了一个interface中的所有方法，我们说类型实现了该interface。

interface变量存储的是实现者的值

```
//1
type I interface{
    Get() int
    Set(int)
}

//2
type S struct{
    Age int
}

func (s S)Get() int{
    return s.Aget
}

func(s *S)Set(age int){
    s.Aget = age
}

//3
func f(i I){
    i.Set(10);
    fmt.Println(i.Get())
}

func main(){
    s := S{}
    f(&s) //
}
```

类型S 实现了接口I的所有方法，因此S是I的实现者，在函数f(i I)中，可以传入I的实现者作为参数。

如何判断interface变量存储的是哪种类型

`value, ok := em.(T)` 其中，em是interface类型变量，T代表要断言的类型，value是interface变量存储的值，ok是bool类型表示是否为该断言的类型T。

```
if t, ok := i.(*S), ok{
    fmt.Println("s implements I", t)
}
```

空的interface

空的interface表示接口没有任何方法，那么所有的类型都可以转换为空interface的，但是go不会对类型是interface{}的slice进行转换，因为interface{}占用的字长跟[]T的长度是不一致的，

interface的实现者的receiver如何选择

在上述例子当中，f(&s) 是按照指针调用的，这里不能通过value调用，因为interface中的Set方法是通过指针调用，当传值时无法进行转换。`go会把指针进行隐式转换得到value，但反过来不行`，即如果把I中的两个方法都通过指针调用，则f(s), f(&s)均可以顺利调用。