C++部分：

封装

封装最好理解了。封装是面向对象的特征之一，是对象和类概念的主要特性。

封装，也就是把客观事物封装成抽象的类，并且类可以把自己的数据和方法只让可信的类或者对象操作，对不可信的进行信息隐藏。

继承

面向对象编程 (OOP) 语言的一个主要功能就是“继承”。继承是指这样一种能力：它可以使用现有类的所有功能，并在无需重新编写原来的类的情况下对这些功能进行扩展。

通过继承创建的新类称为“子类”或“派生类”。

被继承的类称为“基类”、“父类”或“超类”。

继承的过程，就是从一般到特殊的过程。

要实现继承，可以通过“继承”（Inheritance）和“组合”（Composition）来实现。

在某些 OOP 语言中，一个子类可以继承多个基类。但是一般情况下，一个子类只能有一个基类，要实现多重继承，可以通过多级继承来实现。

在考虑使用继承时，有一点需要注意，那就是两个类之间的关系应该是“属于”关系。例如，Employee 是一个人，Manager 也是一个人，因此这两个类都可以继承 Person 类。但是 Leg 类却不能继承 Person 类，因为腿并不是一个人。

抽象类仅定义将由子类创建的一般属性和方法，创建抽象类时，请使用关键字 Interface 而不是 Class。

多态

多态性（polymorphisn）是允许你将父对象设置成为和一个或更多的他的子对象相等的技术，赋值之后，父对象就可以根据当前赋值给它的子对象的特性以不同的方式运作。简单的说，就是一句话：允许将子类类型的指针赋值给父类类型的指针。

实现多态，有二种方式，覆盖，重载。

覆盖，是指子类重新定义父类的虚函数的做法。

重载，是指允许存在多个同名函数，而这些函数的参数表不同（或许参数个数不同，或许参数类型不同，或许两者都不同）。

其实，重载的概念并不属于“面向对象编程”，重载的实现是：编译器根据函数不同的参数表，对同名函数的名称做修饰，然后这些同名函数就成了不同的函数（至少对于编译器来说是这样的）。如，有两个同名函数：function func(p:integer):integer;和function func(p:string):integer;。那么编译器做过修饰后的函数名称可能是这样的：int\_func、str\_func。对于这两个函数的调用，在编译器间就已经确定了，是静态的（记住：是静态）。也就是说，它们的地址在编译期就绑定了（早绑定），因此，重载和多态无关！真正和多态相关的是“覆盖”。当子类重新定义了父类的虚函数后，父类指针根据赋给它的不同的子类指针，动态（记住：是动态！）的调用属于子类的该函数，这样的函数调用在编译期间是无法确定的（调用的子类的虚函数的地址无法给出）。因此，这样的函数地址是在运行期绑定的（晚邦定）。结论就是：重载只是一种语言特性，与多态无关，与面向对象也无关！引用一句Bruce Eckel的话：“不要犯傻，如果它不是晚邦定，它就不是多态。”

那么，多态的作用是什么呢？我们知道，封装可以隐藏实现细节，使得代码模块化；继承可以扩展已存在的代码模块（类）；它们的目的都是为了——代码重用。而多态则是为了实现另一个目的——接口重用！多态的作用，就是为了类在继承和派生的时候，保证使用“家谱”中任一类的实例的某一属性时的正确调用。

虚函数是怎么实现的

虚函数表，动态绑定，

虚函数的实现依赖的是一张**虚函数表（vtable）**，由编译器编译生成并存放在某处，程序运行时在构造对象时会将该地址存放在对象中。在真正调用的时候会先通过存储在对象中的虚函数表的地址，寻找得到真正需要调用的成员函数的地址。

构造函数和析构函数可以是虚函数吗？答案是构造函数不能是虚函数，析构函数可以是虚函数且推荐最好设置为虚函数。

const

static

smart pointer

RAII

使用局部对象来管理资源的技术称为资源获取即初始化；这里的资源主要是指操作系统中有限的东西如内存、网络套接字等等，局部对象是指存储在栈的对象，它的生命周期是由操作系统来管理的，无需人工介入；

敏捷式开发

能用stl绝对不自己写

将smart pointer一定要提到一个思想，就是谁拥有这个指针

太多东西都是私下玩，很多东西在工作中不一定总能碰到。

template平时没用过太野性的方法，都是中规中矩。metaprogramming多少要明白好处在哪。这东西玩好了好厉害，美国在火星上那小车里面几乎都是metaprogramming。

Static linking， dynamic linking， 优缺点

从compile 到likning到最后每一阶段所做的事

Mysql部分：

数据库优化：

**1. 什么是索引?**

索引是一种数据结构,可以帮助我们快速的进行数据的查找.

**2. 索引是个什么样的数据结构呢?**

索引的数据结构和具体存储引擎的实现有关, 在MySQL中使用较多的索引有Hash索引,B+树索引等,而我们经常使用的InnoDB存储引擎的默认索引实现为:B+树索引.

**3. Hash索引和B+树所有有什么区别或者说优劣呢?**

首先要知道Hash索引和B+树索引的底层实现原理:

hash索引底层就是hash表,进行查找时,调用一次hash函数就可以获取到相应的键值,之后进行回表查询获得实际数据.B+树底层实现是多路平衡查找树.对于每一次的查询都是从根节点出发,查找到叶子节点方可以获得所查键值,然后根据查询判断是否需要回表查询数据.

那么可以看出他们有以下的不同:

* hash索引进行等值查询更快(一般情况下),但是却无法进行范围查询.

因为在hash索引中经过hash函数建立索引之后,索引的顺序与原顺序无法保持一致,不能支持范围查询.而B+树的的所有节点皆遵循(左节点小于父节点,右节点大于父节点,多叉树也类似),天然支持范围.

* hash索引不支持使用索引进行排序,原理同上.
* hash索引不支持模糊查询以及多列索引的最左前缀匹配.原理也是因为hash函数的不可预测.**AAAA**和**AAAAB**的索引没有相关性.
* hash索引任何时候都避免不了回表查询数据,而B+树在符合某些条件(聚簇索引,覆盖索引等)的时候可以只通过索引完成查询.
* hash索引虽然在等值查询上较快,但是不稳定.性能不可预测,当某个键值存在大量重复的时候,发生hash碰撞,此时效率可能极差.而B+树的查询效率比较稳定,对于所有的查询都是从根节点到叶子节点,且树的高度较低.

因此,在大多数情况下,直接选择B+树索引可以获得稳定且较好的查询速度.而不需要使用hash索引.

**4. 上面提到了B+树在满足聚簇索引和覆盖索引的时候不需要回表查询数据,什么是聚簇索引?**

在B+树的索引中,叶子节点可能存储了当前的key值,也可能存储了当前的key值以及整行的数据,这就是聚簇索引和非聚簇索引. 在InnoDB中,只有主键索引是聚簇索引,如果没有主键,则挑选一个唯一键建立聚簇索引.如果没有唯一键,则隐式的生成一个键来建立聚簇索引.

当查询使用聚簇索引时,在对应的叶子节点,可以获取到整行数据,因此不用再次进行回表查询.

**5. 非聚簇索引一定会回表查询吗?**

不一定,这涉及到查询语句所要求的字段是否全部命中了索引,如果全部命中了索引,那么就不必再进行回表查询.

举个简单的例子,假设我们在员工表的年龄上建立了索引,那么当进行select age from employee where age < 20的查询时,在索引的叶子节点上,已经包含了age信息,不会再次进行回表查询.

**6. 在建立索引的时候,都有哪些需要考虑的因素呢?**

建立索引的时候一般要考虑到字段的使用频率,经常作为条件进行查询的字段比较适合.如果需要建立联合索引的话,还需要考虑联合索引中的顺序.此外也要考虑其他方面,比如防止过多的所有对表造成太大的压力.这些都和实际的表结构以及查询方式有关.

**7. 联合索引是什么?为什么需要注意联合索引中的顺序?**

MySQL可以使用多个字段同时建立一个索引,叫做联合索引.在联合索引中,如果想要命中索引,需要按照建立索引时的字段顺序挨个使用,否则无法命中索引.

具体原因为:

MySQL使用索引时需要索引有序,假设现在建立了"name,age,school"的联合索引,那么索引的排序为: 先按照name排序,如果name相同,则按照age排序,如果age的值也相等,则按照school进行排序.

当进行查询时,此时索引仅仅按照name严格有序,因此必须首先使用name字段进行等值查询,之后对于匹配到的列而言,其按照age字段严格有序,此时可以使用age字段用做索引查找,,,以此类推.因此在建立联合索引的时候应该注意索引列的顺序,一般情况下,将查询需求频繁或者字段选择性高的列放在前面.此外可以根据特例的查询或者表结构进行单独的调整.

**8. 创建的索引有没有被使用到?或者说怎么才可以知道这条语句运行很慢的原因?**

MySQL提供了explain命令来查看语句的执行计划,MySQL在执行某个语句之前,会将该语句过一遍查询优化器,之后会拿到对语句的分析,也就是执行计划,其中包含了许多信息. 可以通过其中和索引有关的信息来分析是否命中了索引,例如possilbe\_key,key,key\_len等字段,分别说明了此语句可能会使用的索引,实际使用的索引以及使用的索引长度.

**9. 那么在哪些情况下会发生针对该列创建了索引但是在查询的时候并没有使用呢?**

* 使用不等于查询,
* 列参与了数学运算或者函数
* 在字符串like时左边是通配符.类似于'%aaa'.
* 当mysql分析全表扫描比使用索引快的时候不使用索引.
* 当使用联合索引,前面一个条件为范围查询,后面的即使符合最左前缀原则,也无法使用索引.

以上情况,MySQL无法使用索引.

**事务相关**

**1. 什么是事务?**

理解什么是事务最经典的就是转账的栗子,相信大家也都了解,这里就不再说一边了.

事务是一系列的操作,他们要符合ACID特性.最常见的理解就是:事务中的操作要么全部成功,要么全部失败.但是只是这样还不够的.

**2. ACID是什么?可以详细说一下吗?**

A=Atomicity

原子性,就是上面说的,要么全部成功,要么全部失败.不可能只执行一部分操作.

C=Consistency

系统(数据库)总是从一个一致性的状态转移到另一个一致性的状态,不会存在中间状态.

I=Isolation

隔离性: 通常来说:一个事务在完全提交之前,对其他事务是不可见的.注意前面的通常来说加了红色,意味着有例外情况.

D=Durability

持久性,一旦事务提交,那么就永远是这样子了,哪怕系统崩溃也不会影响到这个事务的结果.

**3. 同时有多个事务在进行会怎么样呢?**

多事务的并发进行一般会造成以下几个问题:

* 脏读: A事务读取到了B事务未提交的内容,而B事务后面进行了回滚.
* 不可重复读: 当设置A事务只能读取B事务已经提交的部分,会造成在A事务内的两次查询,结果竟然不一样,因为在此期间B事务进行了提交操作.
* 幻读: A事务读取了一个范围的内容,而同时B事务在此期间插入了一条数据.造成"幻觉".

**4. 怎么解决这些问题呢?MySQL的事务隔离级别了解吗?**

MySQL的四种隔离级别如下:

* 未提交读(READ UNCOMMITTED)

这就是上面所说的例外情况了,这个隔离级别下,其他事务可以看到本事务没有提交的部分修改.因此会造成脏读的问题(读取到了其他事务未提交的部分,而之后该事务进行了回滚).

这个级别的性能没有足够大的优势,但是又有很多的问题,因此很少使用.

* 已提交读(READ COMMITTED)

其他事务只能读取到本事务已经提交的部分.这个隔离级别有 不可重复读的问题,在同一个事务内的两次读取,拿到的结果竟然不一样,因为另外一个事务对数据进行了修改.

* REPEATABLE READ(可重复读)

可重复读隔离级别解决了上面不可重复读的问题(看名字也知道),但是仍然有一个新问题,就是 幻读,当你读取id> 10 的数据行时,对涉及到的所有行加上了读锁,此时例外一个事务新插入了一条id=11的数据,因为是新插入的,所以不会触发上面的锁的排斥,那么进行本事务进行下一次的查询时会发现有一条id=11的数据,而上次的查询操作并没有获取到,再进行插入就会有主键冲突的问题.

* SERIALIZABLE(可串行化)

这是最高的隔离级别,可以解决上面提到的所有问题,因为他强制将所以的操作串行执行,这会导致并发性能极速下降,因此也不是很常用.

**5. Innodb使用的是哪种隔离级别呢?**

InnoDB默认使用的是可重复读隔离级别.

**6. 对MySQL的锁了解吗?**

当数据库有并发事务的时候,可能会产生数据的不一致,这时候需要一些机制来保证访问的次序,锁机制就是这样的一个机制.

就像酒店的房间,如果大家随意进出,就会出现多人抢夺同一个房间的情况,而在房间上装上锁,申请到钥匙的人才可以入住并且将房间锁起来,其他人只有等他使用完毕才可以再次使用.

**7. MySQL都有哪些锁呢?像上面那样子进行锁定岂不是有点阻碍并发效率了?**

从锁的类别上来讲,有共享锁和排他锁.

共享锁: 又叫做读锁. 当用户要进行数据的读取时,对数据加上共享锁.共享锁可以同时加上多个.

排他锁: 又叫做写锁. 当用户要进行数据的写入时,对数据加上排他锁.排他锁只可以加一个,他和其他的排他锁,共享锁都相斥.

用上面的例子来说就是用户的行为有两种,一种是来看房,多个用户一起看房是可以接受的. 一种是真正的入住一晚,在这期间,无论是想入住的还是想看房的都不可以.

锁的粒度取决于具体的存储引擎,InnoDB实现了行级锁,页级锁,表级锁.

他们的加锁开销从大大小,并发能力也是从大到小.

**表结构设计**

**1. 为什么要尽量设定一个主键?**

主键是数据库确保数据行在整张表唯一性的保障,即使业务上本张表没有主键,也建议添加一个自增长的ID列作为主键.设定了主键之后,在后续的删改查的时候可能更加快速以及确保操作数据范围安全.

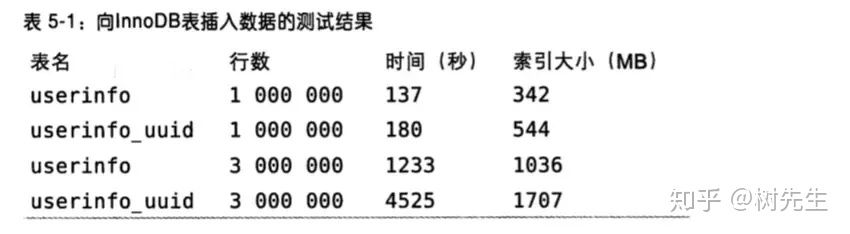
**2. 主键使用自增ID还是UUID?**

推荐使用自增ID,不要使用UUID.

因为在InnoDB存储引擎中,主键索引是作为聚簇索引存在的,也就是说,主键索引的B+树叶子节点上存储了主键索引以及全部的数据(按照顺序),如果主键索引是自增ID,那么只需要不断向后排列即可,如果是UUID,由于到来的ID与原来的大小不确定,会造成非常多的数据插入,数据移动,然后导致产生很多的内存碎片,进而造成插入性能的下降.

总之,在数据量大一些的情况下,用自增主键性能会好一些.

*图片来源于《高性能MySQL》: 其中默认后缀为使用自增ID,\_uuid为使用UUID为主键的测试,测试了插入100w行和300w行的性能.*



关于主键是聚簇索引,如果没有主键,InnoDB会选择一个唯一键来作为聚簇索引,如果没有唯一键,会生成一个隐式的主键.

If you define a PRIMARY KEY on your table, InnoDB uses it as the clustered index.  
If you do not define a PRIMARY KEY for your table, MySQL picks the first UNIQUE index that has only NOT NULL columns as the primary key and InnoDB uses it as the clustered index.

**3. 字段为什么要求定义为not null?**

MySQL官网这样介绍:

NULL columns require additional space in the rowto record whether their values are NULL. For MyISAM tables, each NULL columntakes one bit extra, rounded up to the nearest byte.

null值会占用更多的字节,且会在程序中造成很多与预期不符的情况.

**4. 如果要存储用户的密码散列,应该使用什么字段进行存储?**

密码散列,盐,用户身份证号等固定长度的字符串应该使用char而不是varchar来存储,这样可以节省空间且提高检索效率.

**存储引擎相关**

**1. MySQL支持哪些存储引擎?**

MySQL支持多种存储引擎,比如InnoDB,MyISAM,Memory,Archive等等.在大多数的情况下,直接选择使用InnoDB引擎都是最合适的,InnoDB也是MySQL的默认存储引擎.

1. InnoDB和MyISAM有什么区别?

* InnoDB支持事物，而MyISAM不支持事物
* InnoDB支持行级锁，而MyISAM支持表级锁
* InnoDB支持MVCC, 而MyISAM不支持
* InnoDB支持外键，而MyISAM不支持
* InnoDB不支持全文索引，而MyISAM支持。

**零散问题**

**1. MySQL中的varchar和char有什么区别.**

char是一个定长字段,假如申请了char(10)的空间,那么无论实际存储多少内容.该字段都占用10个字符,而varchar是变长的,也就是说申请的只是最大长度,占用的空间为实际字符长度+1,最后一个字符存储使用了多长的空间.

在检索效率上来讲,char > varchar,因此在使用中,如果确定某个字段的值的长度,可以使用char,否则应该尽量使用varchar.例如存储用户MD5加密后的密码,则应该使用char.

**2. varchar(10)和int(10)代表什么含义?**

varchar的10代表了申请的空间长度,也是可以存储的数据的最大长度,而int的10只是代表了展示的长度,不足10位以0填充.也就是说,int(1)和int(10)所能存储的数字大小以及占用的空间都是相同的,只是在展示时按照长度展示.

**3. MySQL的binlog有有几种录入格式?分别有什么区别?**

有三种格式,statement,row和mixed.

* statement模式下,记录单元为语句.即每一个sql造成的影响会记录.由于sql的执行是有上下文的,因此在保存的时候需要保存相关的信息,同时还有一些使用了函数之类的语句无法被记录复制.
* row级别下,记录单元为每一行的改动,基本是可以全部记下来但是由于很多操作,会导致大量行的改动(比如alter table),因此这种模式的文件保存的信息太多,日志量太大.
* mixed. 一种折中的方案,普通操作使用statement记录,当无法使用statement的时候使用row.

此外,新版的MySQL中对row级别也做了一些优化,当表结构发生变化的时候,会记录语句而不是逐行记录.

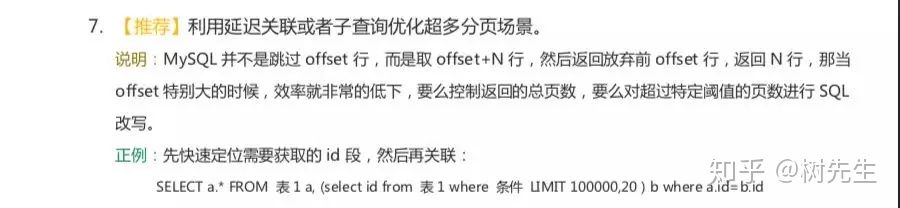
**4. 超大分页怎么处理?**

超大的分页一般从两个方向上来解决.

* 数据库层面,这也是我们主要集中关注的(虽然收效没那么大),类似于select \* from table where age > 20 limit 1000000,10这种查询其实也是有可以优化的余地的. 这条语句需要load1000000数据然后基本上全部丢弃,只取10条当然比较慢. 当时我们可以修改为select \* from table where id in (select id from table where age > 20 limit 1000000,10).这样虽然也load了一百万的数据,但是由于索引覆盖,要查询的所有字段都在索引中,所以速度会很快. 同时如果ID连续的好,我们还可以select \* from table where id > 1000000 limit 10,效率也是不错的,优化的可能性有许多种,但是核心思想都一样,就是减少load的数据.
* 从需求的角度减少这种请求....主要是不做类似的需求(直接跳转到几百万页之后的具体某一页.只允许逐页查看或者按照给定的路线走,这样可预测,可缓存)以及防止ID泄漏且连续被人恶意攻击.

解决超大分页,其实主要是靠缓存,可预测性的提前查到内容,缓存至redis等k-V数据库中,直接返回即可.

在阿里巴巴《Java开发手册》中,对超大分页的解决办法是类似于上面提到的第一种.



**5. 关心过业务系统里面的sql耗时吗?统计过慢查询吗?对慢查询都怎么优化过?**

在业务系统中,除了使用主键进行的查询,其他的我都会在测试库上测试其耗时,慢查询的统计主要由运维在做,会定期将业务中的慢查询反馈给我们.

慢查询的优化首先要搞明白慢的原因是什么? 是查询条件没有命中索引?是load了不需要的数据列?还是数据量太大?

所以优化也是针对这三个方向来的,

* 首先分析语句,看看是否load了额外的数据,可能是查询了多余的行并且抛弃掉了,可能是加载了许多结果中并不需要的列,对语句进行分析以及重写.
* 分析语句的执行计划,然后获得其使用索引的情况,之后修改语句或者修改索引,使得语句可以尽可能的命中索引.
* 如果对语句的优化已经无法进行,可以考虑表中的数据量是否太大,如果是的话可以进行横向或者纵向的分表.

**6. 上面提到横向分表和纵向分表,可以分别举一个适合他们的例子吗?**

横向分表是按行分表.假设我们有一张用户表,主键是自增ID且同时是用户的ID.数据量较大,有1亿多条,那么此时放在一张表里的查询效果就不太理想.我们可以根据主键ID进行分表,无论是按尾号分,或者按ID的区间分都是可以的. 假设按照尾号0-99分为100个表,那么每张表中的数据就仅有100w.这时的查询效率无疑是可以满足要求的.

纵向分表是按列分表.假设我们现在有一张文章表.包含字段id-摘要-内容.而系统中的展示形式是刷新出一个列表,列表中仅包含标题和摘要,当用户点击某篇文章进入详情时才需要正文内容.此时,如果数据量大,将内容这个很大且不经常使用的列放在一起会拖慢原表的查询速度.我们可以将上面的表分为两张.id-摘要,id-内容.当用户点击详情,那主键再来取一次内容即可.而增加的存储量只是很小的主键字段.代价很小.

当然,分表其实和业务的关联度很高,在分表之前一定要做好调研以及benchmark.不要按照自己的猜想盲目操作.

**7. 什么是存储过程？有哪些优缺点？**

存储过程是一些预编译的SQL语句。1、更加直白的理解：存储过程可以说是一个记录集，它是由一些T-SQL语句组成的代码块，这些T-SQL语句代码像一个方法一样实现一些功能（对单表或多表的增删改查），然后再给这个代码块取一个名字，在用到这个功能的时候调用他就行了。2、存储过程是一个预编译的代码块，执行效率比较高,一个存储过程替代大量T\_SQL语句 ，可以降低网络通信量，提高通信速率,可以一定程度上确保数据安全

但是,在互联网项目中,其实是不太推荐存储过程的,比较出名的就是阿里的《Java开发手册》中禁止使用存储过程,我个人的理解是,在互联网项目中,迭代太快,项目的生命周期也比较短,人员流动相比于传统的项目也更加频繁,在这样的情况下,存储过程的管理确实是没有那么方便,同时,复用性也没有写在服务层那么好.

**8. 说一说三个范式**

第一范式: 每个列都不可以再拆分. 第二范式: 非主键列完全依赖于主键,而不能是依赖于主键的一部分. 第三范式: 非主键列只依赖于主键,不依赖于其他非主键.

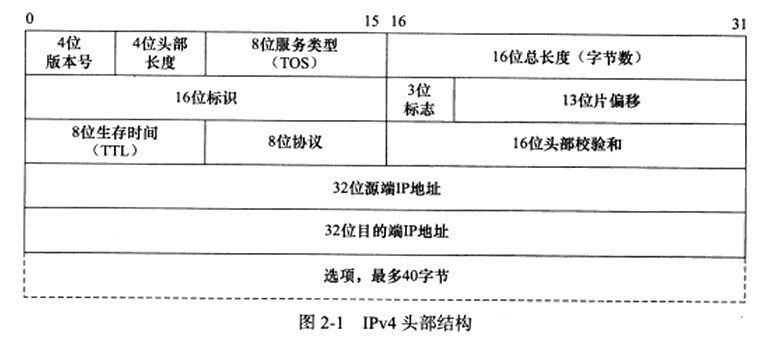
在设计数据库结构的时候,要尽量遵守三范式,如果不遵守,必须有足够的理由.比如性能. 事实上我们经常会为了性能而妥协数据库的设计.

**9. MyBatis中的#和$有什么区别?**

乱入了一个奇怪的问题.....我只是想单独记录一下这个问题,因为出现频率太高了.

----------------------------------------------------------------------------------------------

网络协议部分：



4位版本号指定IP协议版本。

4位头部长度标识该IP头部有多少个32bit（4字节），因为4位最大能表示15（2的n次方减一），所以IP头部最长是60字节。

8位服务类型包括一个3位的优先权字段（现在已经被忽略），4位的TOS字段和1位保留字段（必须置0）。4位的TOS字段分别表示：最小延时，最大吞吐量，最高可靠性和最小费用。其中最多有一个能置为1，应用程序应该根据实际需要来设置它。

16位总长度是指整个IP数据报的长度，以字节为单位，因此IP数据报的最大长度为65535（216-1）字节。但由于MTU的限制，长度超过MTU的数据报都将被分片传输，所以实际传输的IP数据报（或分片）的长度都远远没有达到最大值。接下来的3个字段则描述了如何实现分片。

16位标识唯一地标识主机发送的每一个数据报。其初始值由系统随机生成；每发送一个数据报，其值就加1。该值在数据报分片时被复制到每个分片中，因此同一个数据报的所有分片都具有相同的标识值。

3位标志字段的第一位保留。第二位表示“禁止分片”。如果设置了这个位，IP模块将不对数据报进行分片。在这种情况下，如果IP数据报长度超过MTU的话，IP模块将丢弃该数据报并返回一个ICMP差错报文。第三位表示“更多分片”。除了数据报最后一个分片外，其他分片都要把它置1.

13位分片偏移是分片相对原始IP数据报开始处（仅指数据部分）的偏移。实际的偏移值是该值左移3位（乘8）后得到的。由于这个原因，除了最后一个IP分片外，每个IP的数据部分的长度必须是8的整数倍（这样才能保证后面的IP分片拥有一个合适的偏移值）。

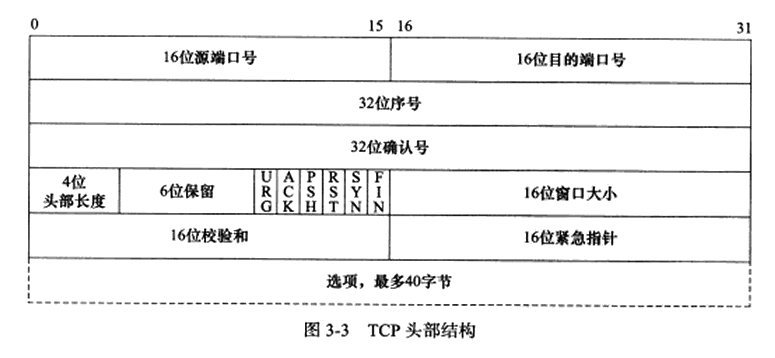
8位生存时间（TTL）是数据报到达目的地之前允许经过的路由器跳数。TTL时间被发送端设置（常见值是64），数据报在转发过程中每经过一个路由，该值就被路由器减1。当TTL值减为0时，路由器将丢弃数据报，并向源端发送一个ICMP差错报文。TTL值可以防止数据报陷入路由循环。

8位协议用来区分上层协议

16位头部校验和由发送端填充，接收端对其使用CRC算法以检验IP数据报头部（注意，仅检验头部）在传输过程中是否损坏。

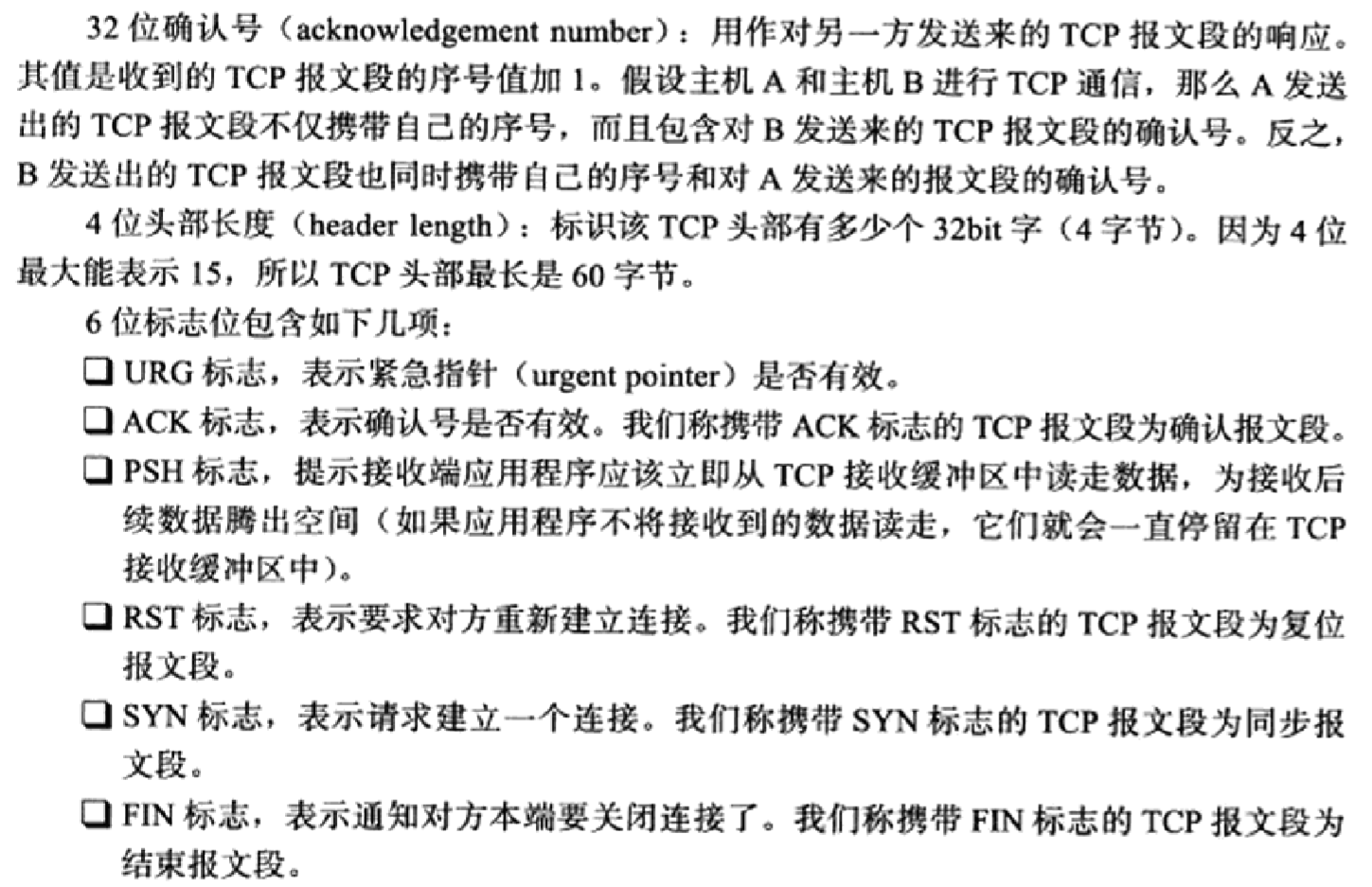
32位的源端IP地址和目的端IP地址用来标识数据报的发送端和接收端。一般情况下，这两个地址在整个数据报的传递过程中保持不变，而不论它中间经过 多少个中转路由器。

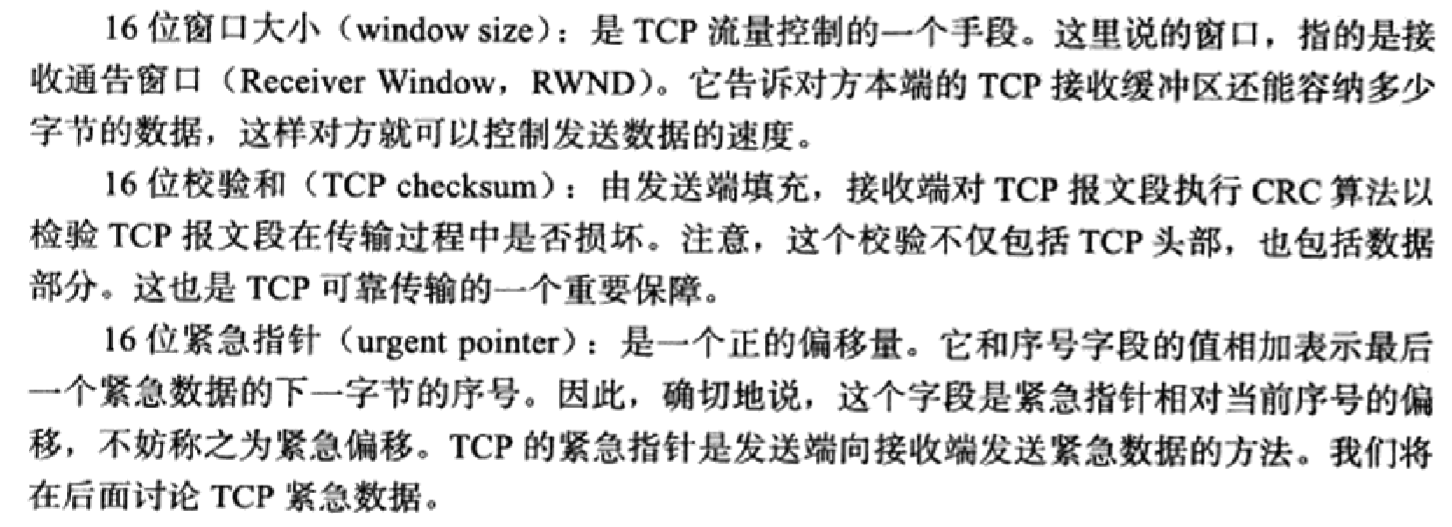
选项字段，是可变长的可选信息。这部分最多包含40字节，因为IP头部最长是60字节（其中还包含前面讨论的20字节的固定部分），可用的IP选项包括：记录路由，时间戳，严格源路由选择

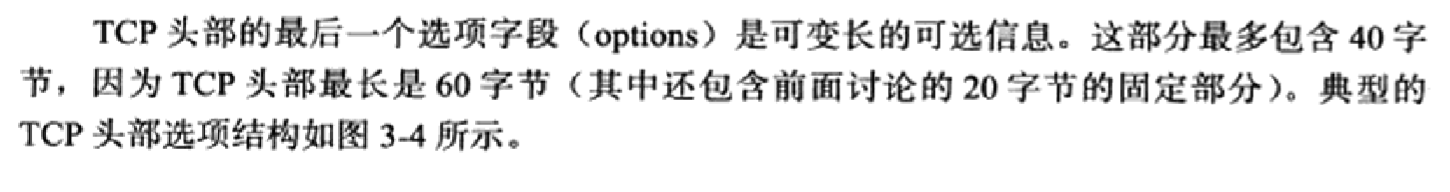


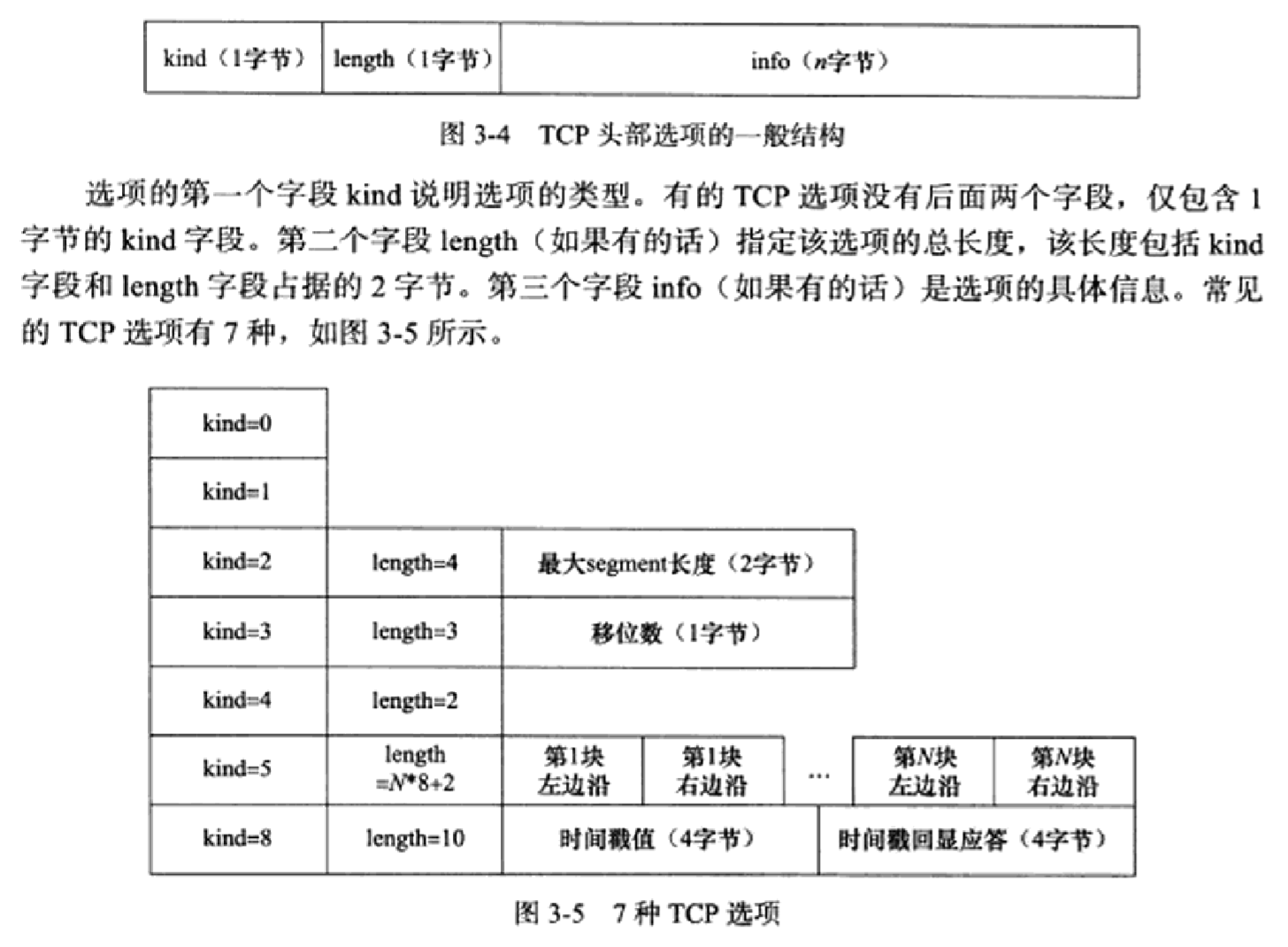
16位端口号：告知主机该报文段来自哪里（源端口）以及传给哪个上层协议或者应用程序（目的端口）的。进行tcp通信时，客户端通常使用系统自动选择的临时端口号，而服务器则使用知名服务端口号。

32位序号：一次TCP通信（从TCP连接建立到断开）过程中某个传输方向上的字节流的每个字节的编号。假设主机a和主机B进行TCP通信，A发送给B的第一个tcp报文段中，序号值被系统初始化为某个随机值ISN。那么在该传输方向上（从a到b），后续的tcp报文段中序号值将被系统设置成ISN加上该报文段所携带数据的第一个字节在整个字节流中的偏移。









## [十分钟搞懂HTTP和HTTPS协议？](https://zhuanlan.zhihu.com/p/72616216)

1.什么是协议？

网络协议是计算机之间为了实现网络通信而达成的一种“约定”或者”规则“，有了这种”约定“，不同厂商的生产设备，以及不同操作系统组成的计算机之间，就可以实现通信。

### 2.HTTP协议是什么？

HTTP协议是**超文本传输协议**的缩写，英文是Hyper Text Transfer Protocol。它是从WEB服务器传输超文本标记语言(HTML)到本地浏览器的传送协议。

设计HTTP最初的目的是为了提供一种发布和接收HTML页面的方法。

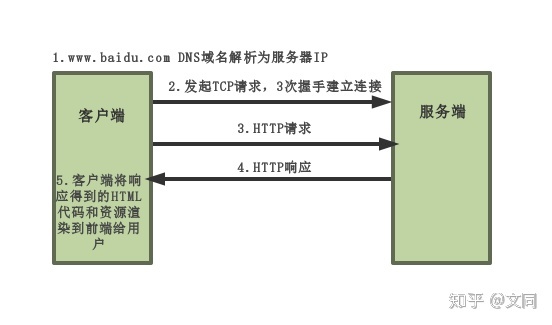
HTPP有多个版本，目前广泛使用的是HTTP/1.1版本。

### 3.HTTP原理

HTTP是一个基于TCP/IP通信协议来传递数据的协议，传输的数据类型为HTML 文件,、图片文件, 查询结果等。

HTTP协议一般用于B/S架构（）。浏览器作为HTTP客户端通过URL向HTTP服务端即WEB服务器发送所有请求。

我们以访问百度为例：

访问百度流程

### 4.HTTP特点

1. http协议支持客户端/服务端模式，也是一种请求/响应模式的协议。
2. 简单快速：客户向服务器请求服务时，只需传送请求方法和路径。请求方法常用的有GET、HEAD、POST。
3. 灵活：HTTP允许传输任意类型的数据对象。传输的类型由Content-Type加以标记。
4. 无连接：限制每次连接只处理一个请求。服务器处理完请求，并收到客户的应答后，即断开连接，但是却不利于客户端与服务器保持会话连接，为了弥补这种不足，产生了两项记录http状态的技术，一个叫做Cookie,一个叫做Session。
5. 无状态：无状态是指协议对于事务处理没有记忆，后续处理需要前面的信息，则必须重传。

### 5.URI和URL的区别

HTTP使用统一资源标识符（Uniform Resource Identifiers, URI）来传输数据和建立连接。

* URI：Uniform Resource Identifier 统一资源**标识**符
* URL：Uniform Resource Location 统一资源**定位**符

URI 是用来标示 一个具体的资源的，我们可以通过 URI 知道一个资源是什么。

URL 则是用来定位具体的资源的，标示了一个具体的资源位置。互联网上的每个文件都有一个唯一的URL。

### 6.HTTP报文组成

### 请求报文构成

1. 请求行：包括请求方法、URL、协议/版本
2. 请求头(Request Header)
3. 请求正文

请求报文组成

### 响应报文构成

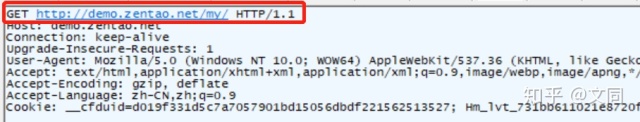
1. 状态行
2. 响应头
3. 响应正文

响应报文组成

### 7.常见请求方法

* GET:请求指定的页面信息，并返回实体主体。
* POST:向指定资源提交数据进行处理请求（例如提交表单或者上传文件）。数据被包含在请求体中。POST请求可能会导致新的资源的建立和/或已有资源的修改。
* HEAD:类似于get请求，只不过返回的响应中没有具体的内容，用于获取报头
* PUT:从客户端向服务器传送的数据取代指定的文档的内容。
* DELETE:请求服务器删除指定的页面。

**get请求**

GET请求

**post请求**

POST请求

**post和get的区别：**

* 都包含请求头请求行，post多了请求body。
* get多用来查询，请求参数放在url中，不会对服务器上的内容产生作用。post用来提交，如把账号密码放入body中。
* GET是直接添加到URL后面的，直接就可以在URL中看到内容，而POST是放在报文内部的，用户无法直接看到。
* GET提交的数据长度是有限制的，因为URL长度有限制，具体的长度限制视浏览器而定。而POST没有。

### 8.响应状态码

**访问一个网页时，浏览器会向web服务器发出请求。此网页所在的服务器会返回一个包含HTTP状态码的信息头用以响应浏览器的请求。**

**状态码分类**：

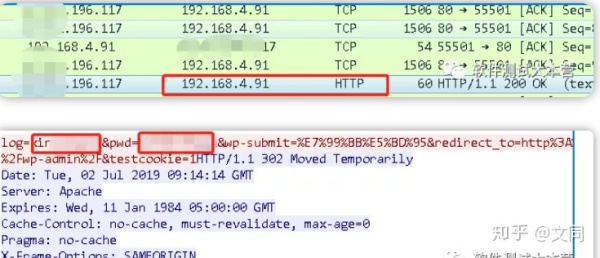
* 1XX- 信息型，服务器收到请求，需要请求者继续操作。
* 2XX- 成功型，请求成功收到，理解并处理。
* 3XX - 重定向，需要进一步的操作以完成请求。
* 4XX - 客户端错误，请求包含语法错误或无法完成请求。
* 5XX - 服务器错误，服务器在处理请求的过程中发生了错误。

**常见状态码**：

* 200 OK - 客户端请求成功
* 301 - 资源（网页等）被永久转移到其它URL
* 302 - 临时跳转
* 400 Bad Request - 客户端请求有语法错误，不能被服务器所理解
* 401 Unauthorized - 请求未经授权，这个状态代码必须和WWW-Authenticate报头域一起使用
* 404 - 请求资源不存在，可能是输入了错误的URL
* 500 - 服务器内部发生了不可预期的错误
* 503 Server Unavailable - 服务器当前不能处理客户端的请求，一段时间后可能恢复正常。

### 9.为什么要用https？

实际使用中，绝大说的网站现在都采用的是https协议，这也是未来互联网发展的趋势。下面是通过wireshark抓取的一个博客网站的登录请求过程。

博客登录抓包

可以看到访问的账号密码都是明文传输， 这样客户端发出的请求很容易被不法分子截取利用，因此，HTTP协议不适合传输一些敏感信息，比如：各种账号、密码等信息，使用http协议传输隐私信息非常不安全。

**一般http中存在如下问题：**

* 请求信息明文传输，容易被窃听截取。
* 数据的完整性未校验，容易被篡改
* 没有验证对方身份，存在冒充危险

### 10.什么是HTTPS?

为了解决上述HTTP存在的问题，就用到了HTTPS。

HTTPS 协议（HyperText Transfer Protocol over Secure Socket Layer）：一般理解为HTTP+SSL/TLS，通过 SSL证书来验证服务器的身份，并为浏览器和服务器之间的通信进行加密。

**那么SSL又是什么？**

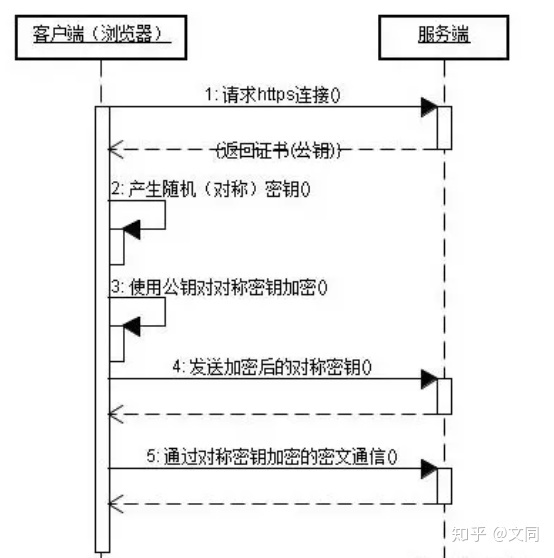
SSL（Secure Socket Layer，安全套接字层）：1994年为 Netscape 所研发，SSL 协议位于 TCP/IP 协议与各种应用层协议之间，为数据通讯提供安全支持。

TLS（Transport Layer Security，传输层安全）：其前身是 SSL，它最初的几个版本（SSL 1.0、SSL 2.0、SSL 3.0）由网景公司开发，1999年从 3.1 开始被 IETF 标准化并改名，发展至今已经有 TLS 1.0、TLS 1.1、TLS 1.2 三个版本。SSL3.0和TLS1.0由于存在安全漏洞，已经很少被使用到。TLS 1.3 改动会比较大，目前还在草案阶段，目前使用最广泛的是TLS 1.1、TLS 1.2。

**SSL发展史（互联网加密通信）**

1. 1994年NetSpace公司设计SSL协议（Secure Sockets Layout）1.0版本，但未发布。
2. 1995年NetSpace发布SSL/2.0版本，很快发现有严重漏洞
3. 1996年发布SSL/3.0版本，得到大规模应用
4. 1999年，发布了SSL升级版TLS/1.0版本，目前应用最广泛的版本
5. 2006年和2008年，发布了TLS/1.1版本和TLS/1.2版本

### 11.浏览器在使用HTTPS传输数据的流程是什么？

HTTPS数据传输流程

1. 首先客户端通过URL访问服务器建立SSL连接。
2. 服务端收到客户端请求后，会将网站支持的证书信息（证书中包含公钥）传送一份给客户端。
3. 客户端的服务器开始协商SSL连接的安全等级，也就是信息加密的等级。
4. 客户端的浏览器根据双方同意的安全等级，建立会话密钥，然后利用网站的公钥将会话密钥加密，并传送给网站。
5. 服务器利用自己的私钥解密出会话密钥。
6. 服务器利用会话密钥加密与客户端之间的通信。

### 12.HTTPS的缺点

* HTTPS协议多次握手，导致页面的加载时间延长近50%；
* HTTPS连接缓存不如HTTP高效，会增加数据开销和功耗；
* 申请SSL证书需要钱，功能越强大的证书费用越高。
* SSL涉及到的安全算法会消耗 CPU 资源，对服务器资源消耗较大。

### 13.总结HTTPS和HTTP的区别

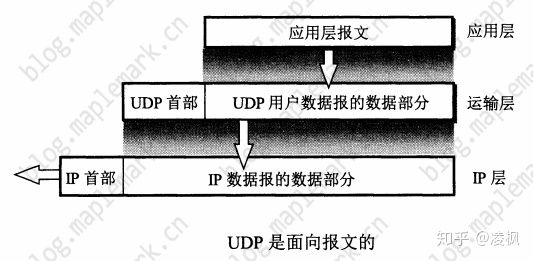
* HTTPS是HTTP协议的安全版本，HTTP协议的数据传输是明文的，是不安全的，HTTPS使用了SSL/TLS协议进行了加密处理。
* http和https使用连接方式不同，默认端口也不一样，http是80，https是443。

## UDP 概述

用户数据报协议 UDP 只在 IP 的数据报服务之上增加了很少一点的功能，这就是复用和分用的功能以及查错检测的功能

### UDP 的主要特点

1. UDP 是无连接的，即发送数据之前不需要建立连接(发送数据结束时也没有连接可释放)，减少了开销和发送数据之前的时延
2. UDP 使用尽最大努力交付，即不保证可靠交付，主机不需要维持复杂的连接状态表
3. UDP 是面向报文的，发送方的 UDP 对应用程序交下来的报文，在添加首部后就向下交付 IP 层。UDP 对应用层交下来的报文，既不合并，也不拆分，而是保留这些报文的边界



1. UDP 没有拥塞控制，网络出现的拥塞不会使源主机的发送速率降低。这对某些实时应用是很重要的
2. UDP 支持一对一、一对多、多对一和多对多的交互通信
3. UDP 的首部开销小，只有8个字节，比 TCP 的20个字节的首部要短

### 存在问题

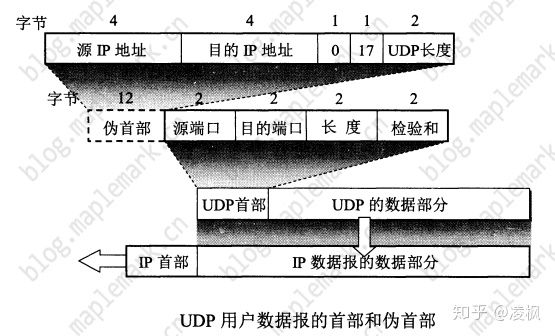
1. 某些实时应用需要使用没有拥塞控制的 UDP，但很多的源主机同时都向网络发送高速率的实时视频流时，网络就有可能发生拥塞，导致大家都无法正常接收。
2. 还有一些使用 UDP 的实时应用，需要对 UDP 的不可靠传输进行适当的改进，以减少数据的丢失。应用进程可以在不影响应用的实时性的前提下，增加一些提高可靠性的措施，如采用前向纠错或重传已丢失的报文

## UDP 的首部格式

用户数据报 UDP 有两个字段：数据字段和首部字段。首部字段很简单，只有8个字节，由四个字段组成，每个字段都是两个字节

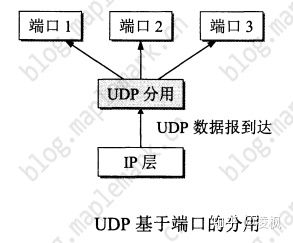
### 首部字段

* 源端口 源端口号。在需要对方回信时。不需要时可用全0
* 目的端口 目的端口号。这在终点交付报文时必须使用
* 长度 UDP 用户数据报的长度，其最小值是8(仅有首部)
* 检验和 检测 UDP 用户数据报在传输中是否有错。有错就丢弃



### 端口分用

当运输层从 IP 层收到 UDP 数据报时，就根据首部中的目的端口，把 UDP 数据报通过相应的端口，上交最后的终点——应用进程



如果接受方 UDP 发现收到的报文中的目的端口号不正确(即不存在对应于该端口号的应用程序)，就丢弃该报文，并由网际控制报文协议 ICMP 发送“端口不可达”差错报文给发送方

### 伪首部

UDP 用户数据报首部中检验和的计算方法有些特殊。在计算检验和时，要在 UDP 用户数据报之前增加 12 个字节的伪首部。所谓“伪首部”是因为这种伪首部并不是 UDP 用户数据报真正的首部。只是在计算检验和时，临时添加在 UDP 用户数据报前面，得到一个临时的 UDP 用户数据报。检验和就是按照这个临时用户数据报来计算的。伪首部既不向下传也不向上递交，而仅仅是为了计算检验和

---------------------------------------------------------------------------------------------------

设计模式

-----------------------------------------------------------------------------------------------------

STL