

# Spring

---

```
1  @Autowired
2  -可以写在属性上，此时不需要set方法
3  -可以写在set方法上
4  默认ByType注入，可以和@Qualifier注解一同使用，ByName注入
5  @Resource
6  默认ByName注入，可以指定name或type分别按照名称和类型进行注入
7
8  @Primary 必须和@Component等注解一同使用，也即bean.xml文件中配置了，类上不
   加@Component的话无法使用
```

# MyBatis

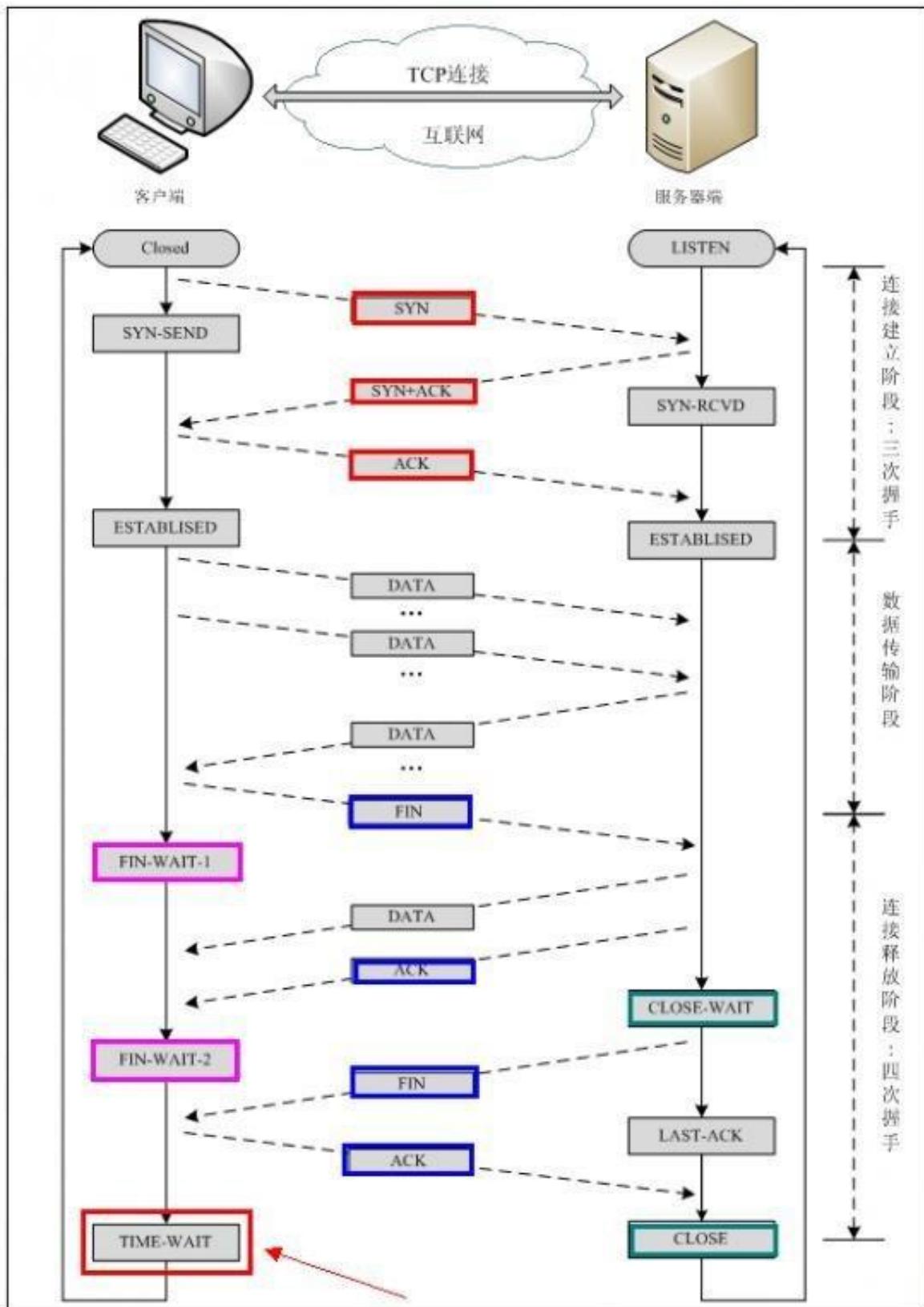
---

# 计算机网络

---

## 1.TCP三次握手和四次挥手

---



## 2.time-wait的作用

time-wait开始的时间为tcp四次挥手中主动关闭连接方发送完最后一次挥手，也就是ACK=1的信号结束后，主动关闭连接方所处的状态。（即发送端最后一次挥手后等待2msl）

然后time-wait的持续时间为2MSL. MSL是Maximum Segment Lifetime,译为“报文最大生存时间”，可为30s, 1min或2min。2msl就是2倍的这个时间。工程上为2min，2msl就是4min。但一般根据实际的网络情况进行确定。

原因1：为了保证客户端发送的最后一个ack报文段能够到达服务器。因为这最后一个ack确认包可能会丢失，然后服务器就会超时重传第三次挥手的fin信息报，然后客户端再重传一次第四次挥手的ack报文。如果没有这2msl，客户端发送完最后一个ack数据报后直接关闭连接，那么就接收不到服务器超时重传的fin信息报(此处应该是客户端收到一个非法的报文段，而返回一个RST的数据报，表明拒绝此次通信，然后双方就产生异常，而不是收不到。)，那么服务器就不能按正常步骤进入close状态。那么就会耗费服务器的资源。当网络中存在大量的timewait状态(高并发短连接时)，那么服务器的压力可想而知。

原因2：在第四次挥手后，经过2msl的时间足以让本次连接产生的所有报文段都从网络中消失，这样下一次新的连接中就肯定不会出现旧连接的报文段了。也就是防止我们上一篇文章[为什么tcp是三次握手而不是两次握手？](#)中说的：已经失效的连接请求报文段出现在本次连接中。如果没有的话就可能这样：这次连接一挥手完马上就结束了，没有timewait。这次连接中有个迷失在网络中的syn包，然后下次连接又马上开始，下个连接发送syn包，迷失的syn包忽然又到达了对面，所以对面可能同时收到或者不同时间收到请求连接的syn包，然后就出现问题了。

### 3.为什么TCP是三次握手？而不是两次或四次？

初始

	服务端发送	服务端接收	客户端发送	客户端接收
服务端	○	○	○	○
客户端	○	○	○	○

第一次握手

客户端向服务端发送连接请求报文段。该报文段的头部中SYN=1，ACK=0，seq=x。服务端接收到报文后，可以确认客户端的发送功能，服务端的接收功能，都是正常的。

	服务端发送	服务端接收	客户端发送	客户端接收
服务端	○	√	√	○
客户端	○	○	○	○

第二次握手

服务端收到连接请求报文段后，如果同意连接，则会发送一个应答：SYN=1，ACK=1，seq=y，ack=x+1。客户端接收到第二次握手报文后，可以确认服务端的发送功能，客户端的接收功能是正常的。不过，此时服务端并不知道自己的发送功能，客户端的接收功能是否正常。

	服务端发送	服务端接收	客户端发送	客户端接收
服务端	○	√	√	○
客户端	√	√	√	√

第三次握手

当客户端收到连接同意的应答后，还要向服务端发送一个确认报文段，表示：服务端发来的连接同意应答已经成功收到。该报文段的头部为：ACK=1，seq=x+1，ack=y+1 此时，服务端就可以确认自己的发送功能，客户端的接收功能是正常的。

	服务端发送	服务端接收	客户端发送	客户端接收
服务端	√	√	√	√
客户端	√	√	√	√

## 4.如果已经建立了连接，但是客户端突然出现故障了怎么办？

TCP还有一个保活计时器，显然，客户端如果出现故障，服务器不能一直等下去，白白浪费资源。服务器每收到一次客户端的请求后都会重新复位这个计时器，时间通常是设置为2小时，若两小时还没有收到客户端的任何数据，服务器就会发送一个探测报文段，以后每隔75秒钟发送一次。若一连发送10个探测报文仍然没反应，服务器就认为客户端出了故障，接着就关闭连接。

## 5.TCP与UDP的区别

### 1. 对比

	UDP	TCP
是否连接	无连接	面向连接
是否可靠	不可靠传输，不使用流量控制和拥塞控制	可靠传输，使用流量控制和拥塞控制
连接对象个数	支持一对一，一对多，多对一和多对多交互通信	只能是一对一通信
传输方式	面向报文	面向字节流
首部开销	首部开销小，仅8字节	首部最小20字节，最大60字节
适用场景	适用于实时应用（IP电话、视频会议、直播等）	适用于要求可靠传输的应用，例如文件传输

### 2. 总结

- TCP向上层提供面向连接的可靠服务，UDP向上层提供无连接不可靠服务。
- 虽然UDP并没有TCP传输来的准确，但是也能在很多实时性要求高的地方有所作为
- 对数据准确性要求高，速度可以相对较慢的，可以选用TCP

# 操作系统

---

## java基础

---

### 1.Int 和Integer的比较

---

```
1 Integer i = 100;
2 Integer j = 100;
3 System.out.print(i == j); //true
4
5 Integer i = 128;
6 Integer j = 128;
7 System.out.print(i == j); //false
```

java在编译Integer i = 100 ;时, 会翻译成为Integer i = Integer.valueOf(100)。而java API中对Integer类型的valueOf的定义如下, 对于-128到127之间的数, 会进行缓存, Integer i = 127时, 会将127这个Integer对象进行缓存, 下次再写Integer j = 127时, 就会直接从缓存中取, 就不会new了。

### 2.接口和抽象类的区别

---

1.接口的方法默认是 public,所有方法在接口中不能有实现(Java8开始接口方法可以有默认实现),而抽象类可以有非抽象的方法。2.接口中除了 static、final变量,不能有其他变量,而抽象类中则不一定。3.一个类可以实现多个接口,但只能实现一个抽象类。接口自己本身可以通过 extends关键字扩展多个接口。4.接口方法默认修饰符是 public,抽象方法可以有 public、protected和 default这些修饰符(抽象方法就是为了被重写所以不能使用private关键字修饰! 5.从设计层面来说,抽象是对类的抽象,是一种模板设计,而接口是对行为的抽象,是一种行为的规范。

总结一下jdk1.7~jdk1.9java中接口概念的变化: 1.在jdk7或更早版本中,接口里面只能有常量变量和抽象方法。这些接口方法必须由选择实现接口的类实现。2.jdk8的时候接口可以有默认方法和静态方法功能。3.jdk9在接口中引入了私有方法和私有静态方法。

### 3.深拷贝与浅拷贝

---

[Java 浅拷贝和深拷贝的理解和实现方式](#)

## java集合

---

## java多线程

---

## java虚拟机

---

# 1.ThreadLocal 的内存泄漏问题

[ThreadLocal 内存泄漏问题深入分析](#)

## 数据库

### 1.MyISAM和InnoDB区别和应用场景

#### 概述

**MyISAM**：它是基于传统的ISAM类型，ISAM是Indexed Sequential Access Method (有索引的顺序访问方法) 的缩写，它是存储记录 and 文件的标准方法。不是事务安全的，而且不支持外键，如果执行大量的 select, insert MyISAM比较适合。

**InnoDB**：支持事务安全的引擎，支持外键、行锁、事务是他的最大特点。如果有大量的 update 和 insert，建议使用InnoDB，特别是针对多个并发和QPS较高的情况。

	InnoDB	MyIsam
存储文件	.frm 表定义文件 .idb 数据文件	.frm 表定义文件 .myd 数据文件 .myi 索引文件
支持的锁	支持行锁和表锁	支持表锁
事物	事物型索引	不支持事物
count	扫表	专门存储地方
索引结构	B+tree索引	B+tree索引

#### MyISAM与InnoDB的主要区别:

##### 存储结构

MyISAM：每个MyISAM在磁盘上存储成三个文件。1) .frm 用于存储表的定义。2) .MYD 用于存放数据。3) .MYI 用于存放表索引

InnoDB：所有的表都保存在同一个数据文件中（也可能是多个文件，或者是独立的表空间文件），InnoDB表的大小只受限于操作系统文件的大小，一般为2GB。

##### 存储空间

MyISAM：可被压缩，存储空间较小。支持三种不同的存储格式：静态表(默认，但是注意数据末尾不能有空格，会被去掉)、动态表、压缩表。

InnoDB：需要更多的内存和存储，它会在主内存中建立其专用的缓冲池用于高速缓冲数据和索引。

##### 事务支持

MyISAM：强调的是性能，每次查询具有原子性,其执行数度比InnoDB类型更快，但是不提供事务支持。  
InnoDB：提供事务支持事务，外部键等高级数据库功能。具有事务(commit)、回滚(rollback)和崩溃修复能力(crash recovery capabilities)的事务安全(transaction-safe (ACID compliant))型表。

### 表主键

MyISAM：允许没有任何索引和主键的表存在，索引都是保存行的地址。InnoDB：如果没有设定主键或者非空唯一索引，就会自动生成一个6字节的主键(用户不可见)，数据是主索引的一部分，附加索引保存的是主索引的值。

## 应用场景

1. **MyISAM**管理非事务表。它提供高速存储和检索，以及全文搜索能力。如果应用中需要执行大量的SELECT查询，那么MyISAM是更好的选择。
2. **InnoDB**用于事务处理应用程序，具有众多特性，包括ACID事务支持。如果应用中需要执行大量的INSERT或UPDATE操作，则应该使用InnoDB，这样可以提高多用户并发操作的性能。

在大数据量，高并发量的互联网业务场景下，请使用InnoDB: 行锁，对提高并发帮助很大，事务，对数据一致性帮助很大这两个点，是InnoDB最吸引人的地方。

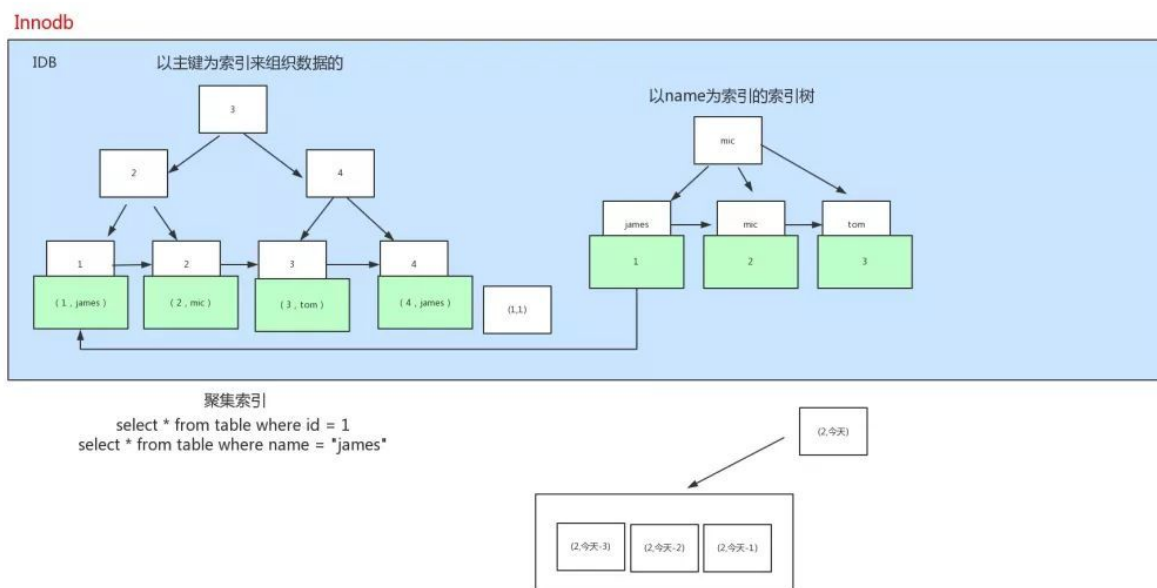
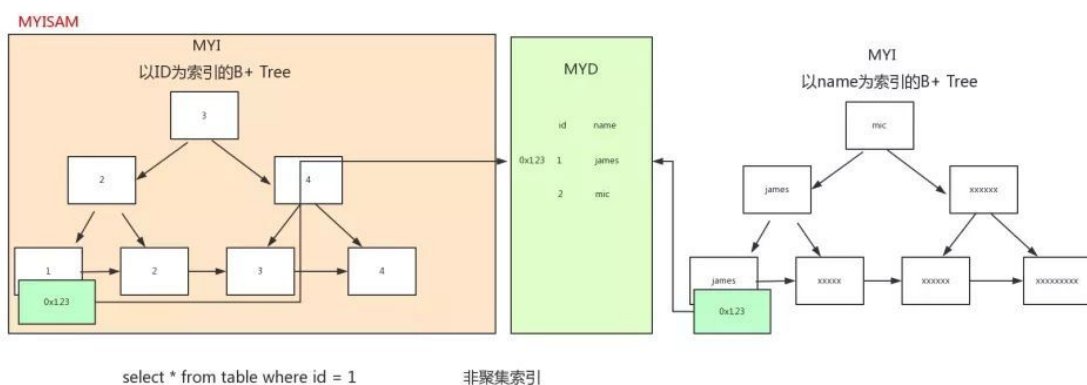
[深入理解MySQL索引底层实现原理 | 技术干货](#)

[mysql索引底层原理](#)

## 2.MySQL 中 MyISAM 中的查询为什么比 InnoDB 快?

---





那么为什么大家喜欢说 **MyisAM** 查询快呢？那是因为，**InnoDB** 的表是根据主键进行展开的 B+tree 的 **聚集索引**。**MyIsam** 则**非聚集型索引**，myisam 存储会有两个文件，一个是索引文件，另外一个 是数据文件，其中索引文件中的索引指向数据文件中的表数据。

聚集型索引并不是一种单独的索引类型，而是一种存储方式，InnoDB 聚集型索引实际上是在同一结构中保存了 B+tree 索引和数据行。当有聚簇索引时，它的索引实际放在叶子页中。

结合图，可以看出：**INNODB** 在做 **SELECT** 的时候，要维护的东西比 **MYISAM** 引擎多很多。

1. 数据块，INNODB 要缓存，MYISAM 只缓存索引块，这中间还有换进换出的减少；
2. innodb 寻址要映射到块，再到行，MYISAM 记录的直接是文件的 OFFSET，定位比 INNODB 要快
3. INNODB 还需要维护 MVCC 一致；虽然你的场景没有，但他还是需要去检查和维护 MVCC (Multi-Version Concurrency Control) 多版本并发控制。


微信号: yucao

InnoDB：通过为每一行记录添加两个额外的隐藏的值来实现 **MVCC**，这两个值一个记录这行数据何时被创建，另外一个记录这行数据何时过期（或者被删除）。但是 InnoDB 并不存储这些事件发生时的实际时间，相反它只存储这些事件发生时的系统版本号。这是一个随着事务的创建而不断增长的数字。每个事务在事务开始时会记录它自己的系统版本号。每个查询必须去检查每行数据的版本号与事务的版本号是否相同。让我们来看看当隔离级别是 REPEATABLE READ 时这种策略是如何应用到特定的操作的：

SELECT InnoDB 必须每行数据来保证它符合两个条件：



1. InnoDB 必须找到一个行的版本，它至少要和事务的版本一样老(也即它的版本号不大于事务的版本号)。这保证了不管是事务开始之前，或者事务创建时，或者修改了这行数据的时候，这行数据是存在的。
2. 这行数据的删除版本必须是未定义的或者比事务版本要大。这可以保证在事务开始之前这行数据没有被删除。

 微信号: yyucaao

### 3.索引失效的情况

[一张图搞懂MySQL的索引失效](#)

[Mysql索引\\_\(一篇就够le\)\\_](#)

### 4.sql注入的原理及防范

#### 原理

sql注入的原理是**将sql代码伪装到输入参数中，传递到服务器解析并执行**的一种攻击手法。也就是说，在一些对server端发起的请求参数中植入一些sql代码，server端在执行sql操作时，会拼接对应参数，同时也将一些sql注入攻击的“sql”拼接起来，导致会执行一些预期之外的操作。

#### 示例：

比如我们使用的登录接口：在登录界面包括用户名和密码输入框，以及提交按钮，输入用户名和密码，提交。

登录时调用接口/user/login/ 加上参数username、password，首先连接数据库，然后后台对请求参数中携带的用户名、密码进行参数校验，即sql的查询过程。假设正确的用户名和密码为ls和123456，输入正确的用户名和密码、提交，相当于调用了以下的SQL语句。

```
1 | SELECT * FROM user WHERE username = 'ls' AND password = '123456'
```

sql中会将#及--以后的字符串当做注释处理，如果我们使用“' or 1=1 #'”作为用户名参数，那么服务端构建的sql语句就如下：

```
1 | select * from users where username=' ' or 1=1#' and password='123456'
```

1=1属于常等型条件，因此这个sql便成为了如下，查询出所有的登陆用户。

```
1 | select * from users
```

其实上面的sql注入只是在参数层面做了些手脚，如果是引入了一些功能性的sql那就更危险了，比如上面的登陆接口，如果用户名使用这个“' or 1=1;delete \* from users; #”，那么在";"之后相当于是另外一条新的sql，这个sql是删除全表，是非常危险的操作，因此sql注入这种还是需要特别注意的。

#### 防范---sql预编译

在知道了sql注入的原理之后，我们同样也了解到mysql有预编译的功能，指的是在服务器启动时，mysql client把sql语句的模板（变量采用占位符进行占位）发送给mysql服务器，mysql服务器对sql语句的模板进行编译，编译之后根据语句的优化分析对相应的索引进行优化，在最终绑定参数时把相应的参数传递给mysql服务器，直接进行执行，节省了sql查询时间，以及mysql服务器的资源，达到一次编译、多次执行的目的，除此之外，还可以防止SQL注入。

```
1 #jdbc中
2 String sql = "select id, no from user where id=?";
3 PreparedStatement ps = conn.prepareStatement(sql);
4 ps.setInt(1, id);
5 ps.executeQuery();
```

## MyBatis中的防止

#将传入的数据都当成一个字符串，会对自动传入的数据加一个双引号。如: where username=#{username}，如果传入的值是111,那么解析成sql时的值为where username="111", 如果传入的值是id，则解析成的sql为where username="id".

\$将传入的数据直接显示在sql中

```
1 <select id="selectByNameAndPassword" parameterType="java.util.Map"
  resultMap="BaseResultMap">
2   select id, username, password, role
3   from user
4   where username = #{username,jdbcType=VARCHAR}
5   and password = #{password,jdbcType=VARCHAR}
6 </select>#可以防止sql注入
7
8 <select id="selectByNameAndPassword" parameterType="java.util.Map"
  resultMap="BaseResultMap">
9   select id, username, password, role
10  from user
11  where username = ${username,jdbcType=VARCHAR}
12  and password = ${password,jdbcType=VARCHAR}
13 </select>#不可以防止sql注入
```

#{ }是经过预编译的，是安全的；\${ }是未经过预编译的，仅仅是取变量的值，是非安全的，存在SQL注入。

MySQL框架下SQL注入的三种攻击方式：其一

## in 之后的多个参数

in之后多个id查询时使用# 同样会报错，

```
1 Select * from news where id in (#{ids})
```

正确用法为使用foreach，而不是将#替换为\$

```
1 id in
2 <foreach collection="ids" item="item" open="(" separator="," close=")">
3   #{ids}
4 </foreach>
```

[Mybatis 框架下 SQL 注入攻击的 3 种方式, 真是防不胜防!](#)

## 5.MySQL实现分页查询

### limit 基本实现方式

一般情况下，客户端通过传递 pageNo（页码）、pageSize（每页条数）两个参数去分页查询数据库中的数据，在数据量较小（元组百/千级）时使用 MySQL自带的 `limit` 来解决这个问题：

收到客户端{pageNo:1,pageSize:10} `select * from table limit (pageNo-1)*pageSize, pageSize;`

收到客户端{pageNo:5,pageSize:30} `select * from table limit (pageNo-1)*pageSize, pageSize;`

### 建立主键或者唯一索引

在数据量较小的时候简单的使用 `limit` 进行数据分页在性能上面不会有明显的缓慢，但是数据量达到了 **万级到百万级** sql语句的性能将会影响数据的返回。这时需要利用主键或者唯一索引进行数据分页；

假设主键或者唯一索引为 good\_id 收到客户端{pageNo:5,pageSize:10} `select * from table where good_id > (pageNo-1)*pageSize limit pageSize; --返回good_id为40到50之间的数据`

### 基于数据再排序

当需要返回的信息为顺序或者倒序时，对上面的语句基于数据再排序。order by ASC/DESC 顺序或倒序默认为顺序

`select * from table where good_id > (pageNo-1)*pageSize order by good_id limit pageSize; --返回good_id为40到50之间的数据,数据依据good_id顺序排列`

## 6.B树与B+树的比较

1、**B+树的层级更少**：相较于B树B+每个非叶子节点存储的关键字数更多，树的层级更少所以查询数据更快；

2、**B+树查询速度更稳定**：B+所有关键字数据地址都存在叶子节点上，所以每次查找的次数都相同所以查询速度要比B树更稳定；

3、**B+树天然具备排序功能**：B+树所有的叶子节点数据构成了一个有序链表，在查询大小区间的数据时候更方便，数据紧密性很高，缓存的命中率也会比B树高。

4、B+树全节点遍历更快：B+树遍历整棵树只需要遍历所有的叶子节点即可，，而不需要像B树一样需要对每一层进行遍历，这有利于数据库做全表扫描。

B树相对于B+树的优点是，如果经常访问的数据离根节点很近，而B树的非叶子节点本身存有关键字其数据的地址，所以这种数据检索的时候会要比B+树快。

## 7.MySQL唯一索引的作用和使用场景

### 普通索引

这是最基本的索引类型，而且它没有唯一性之类的限制。

### 唯一性索引

这种索引和前面的“普通索引”基本相同，但有一个区别：索引列的所有值都只能出现一次，即必须唯一。

这两种索引的运行原理

### 查询过程

对于普通索引来说，查找到满足条件的第一个记录后，需要查找下一个记录，直到碰到第一个不满足条件的记录。

对于唯一索引来说，由于索引定义了唯一性，查找到第一个满足条件的记录后，就会停止继续检索。

所以在这里你感觉用唯一性索引会快一些，毕竟少了一个步骤。但是这个不同带来的性能差距微乎其微。

你知道的，InnoDB 的数据是按数据页为单位来读写的。也就是说，当需要读一条记录的时候，并不是将这个记录本身从磁盘读出来，而是以页为单位，将其整体读入内存。在 InnoDB 中，每个数据页的大小默认是 16KB。

因为引擎是按页读写的，所以说，当找到符合条件的记录的时候，它所在的数据页就都在内存里了。那么，对于普通索引来说，要多做的那一次“查找和判断下一条记录”的操作，就只需要一次指针寻找和一次计算。

当然也会有特殊情况，就是符合条件的记录正好处于数据页的最后一个，那往下查找的操作就会拿下一个数据页放进内存，这个时候就会慢了，但是一个整型字段，一个数据页可以放进千的key，所以这个概率很低。

### 索引具体的处理流程

清楚了change buffer然后模拟一个场景来看一下两种索引具体的处理流程是怎样的。

如果要在这张表中插入一个 id=5 的新纪录，InnoDB 的处理流程是怎样的。

**第一种情况：目标数据页在内存中。**

唯一索引：找到 4 和 6 之间的位置，判断到没有冲突，插入这个值，语句执行结束；普通索引：找到 4 和 6 之间的位置，插入这个值，语句执行结束。这样看来，普通索引和唯一索引对更新语句性能影响的差别，只是一个判断，只会耗费微小的 CPU 时间。

**第二种情况是：目标数据页不在内存中**

唯一索引：需要将数据页读入内存，判断到没有冲突，插入这个值，语句执行结束；普通索引：则是将更新记录在 change buffer，语句执行就结束了。主要区别就是唯一索引需要把磁盘中的数据页放入内存。就是这步影响了性能。

将数据从磁盘读入内存涉及随机 IO 的访问，是数据库里面成本最高的操作之一。change buffer 因为减少了随机磁盘访问，所以对更新性能的提升是会很明显的。

但是普通索引用 change buffer 起到加速作用也是有应用场景的。

因为 merge 的时候是真正进行数据更新的时刻，而 change buffer 的主要目的就是记录的变更动作缓存下来，所以在数据页做 merge 之前，change buffer 记录的变更越多（也就是这个页面上要更新的次数越多），收益就越大。

由此看来就是对于写多读少的业务来说，页面在写完以后马上被访问到的概率比较小，此时 change buffer 的使用效果最好。这种业务模型常见的就是账单类、日志类的系统。

所以反过来，假设一个业务的更新模式是写入之后马上会做查询，那么即使满足了条件，将更新先记录在 change buffer，但之后由于马上要访问这个数据页，会立即触发 merge 过程。这样随机访问 IO 的次数不会减少，反而增加了 change buffer 的维护代价。所以，对于这种业务模式来说，change buffer 反而起到了副作用。

#### 索引使用选择

从上面的内容来说，这两类索引在查询能力上是没差别的，主要考虑的是对更新性能的影响。所以，我建议你尽量选择普通索引。如果所有的更新后面，都马上伴随着对这个记录的查询，那么你应该关闭 change buffer。而在其他情况下，change buffer 都能提升更新性能。

在实际使用中，你会发现，普通索引和 change buffer 的配合使用，对于数据量大的表的更新优化还是很明显的。特别地，在使用机械硬盘时，change buffer 这个机制的收效是非常显著的。所以，当你有一个类似“历史数据”的库，并且出于成本考虑用的是机械硬盘时，那你应该特别关注这些表里的索引，尽量使用普通索引，然后把 change buffer 尽量开大，以确保这个“历史数据”表的数据写入速度。这时候，归档数据已经是确保没有唯一键冲突了。要提高归档效率，可以考虑把表里面的唯一索引改成普通索引。

唯一索引使用的问题，主要是纠结在“业务可能无法确保”的情况。首先，业务正确性优先。如果业务不能保证数据的唯一性，或者业务就是要求数据库来做约束，那么没得选，必须创建唯一索引。本篇文章的意义在于，如果碰上了大量插入数据慢、内存命中率低的时候，可以给你多提供一个排查思路。

## 8. 内连接、左外连接、右外连接、全外连接的区别

#### 基本定义：

left join（左连接）：返回包括左表中的所有记录和右表中连接字段相等的记录。

right join（右连接）：返回包括右表中的所有记录和左表中连接字段相等的记录。

inner join（等值连接或者叫内连接）：只返回两个表中连接字段相等的行。

full join（全外连接）：返回左右表中所有的记录和左右表中连接字段相等的记录。

1	A表
2	id name
3	1 小王

```

4      2    小李
5      3    小刘
6      B表
7      id    A_id    job
8      1     2      老师
9      2     4      程序员
10     #内连接
11     select a.name,b.job from A a inner join B b on a.id=b.A_id
12     只能得到一条记录
13     小李    老师
14     #左外连接 左表不加限制
15     select a.name,b.job from A a left join B b on a.id=b.A_id
16     三条记录
17     小王    null
18     小李    老师
19     小刘    null
20     #右外连接 右表不加限制
21     select a.name,b.job from A a right join B b on a.id=b.A_id
22     两条记录
23     小李    老师
24     null    程序员
25     #全连接，在内连接的基础上 左右两表都不加限制
26     select a.name,b.job from A a full join B b on a.id=b.A_id
27     四条数据
28     小王    null
29     小李    老师
30     小刘    null
31     null    程序员

```

## 9.悲观锁、乐观锁

[【BAT面试题系列】面试官：你了解乐观锁和悲观锁吗？](#)

## http

### 1.http1.0 http1.1 http2.0的区别





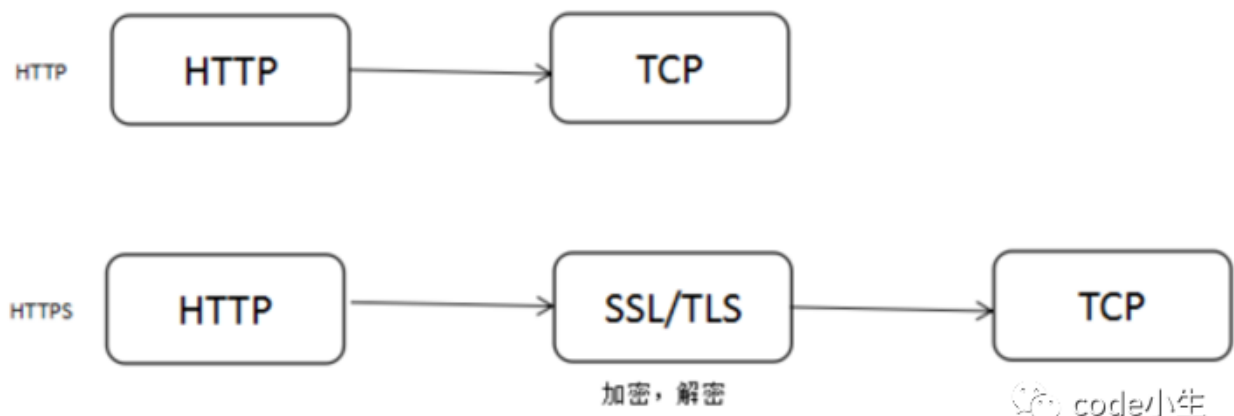
## HTTP1.0和HTTP1.1的一些区别

HTTP1.0最早在网页中使用是在1996年，那个时候只是使用一些较为简单的网页上和网络请求上，而HTTP1.1则在1999年才开始广泛应用于现在的各大浏览器网络请求中，同时HTTP1.1也是当前使用最为广泛的HTTP协议。主要区别主要体现在：

1. **缓存处理**，在HTTP1.0中主要使用header里的If-Modified-Since,Expires来做为缓存判断的标准，HTTP1.1则引入了更多的缓存控制策略例如Entity tag, If-Unmodified-Since, If-Match, If-None-Match等更多可供选择的缓存头来控制缓存策略。
2. **带宽优化及网络连接的使用**，HTTP1.0中，存在一些浪费带宽的现象，例如客户端只是需要某个对象的一部分，而服务器却将整个对象送过来了，并且不支持断点续传功能，HTTP1.1则在请求头引入了**range头域**，它允许只请求资源的某个部分，即**返回码是206**（Partial Content），这样就方便了开发者自由的选择以便于充分利用带宽和连接。
3. **错误通知的管理**，在HTTP1.1中新增了**24个错误状态响应码**，如**409**（Conflict）表示请求的资源与资源的当前状态发生冲突；**410**（Gone）表示服务器上的某个资源被永久性的删除。
4. **Host头处理**，在HTTP1.0中认为每台服务器都绑定一个唯一的IP地址，因此，请求消息中的URL并没有传递主机名（hostname）。但随着虚拟主机技术的发展，在**一台物理服务器上可以存在多个虚拟主机**（Multi-homed Web Servers），并且它们共享一个IP地址。HTTP1.1的请求消息和响应消息都应**支持Host头域**，且请求消息中如果没有Host头域会报告一个错误（400 Bad Request）。
5. **长连接**，HTTP 1.1支持长连接（Persistent Connection）和请求的流水线（Pipelining）处理，在**一个TCP连接上可以传送多个HTTP请求和响应**，减少了建立和关闭连接的消耗和延迟，在HTTP1.1中默认开启Connection: **keep-alive**，一定程度上弥补了HTTP1.0每次请求都要创建连接的缺点。

## HTTPS与HTTP的一些区别

- HTTPS协议需要到**CA申请证书**，一般免费证书很少，需要交费。
- HTTP协议运行在TCP之上，所有传输的内容都是**明文**，**HTTPS运行在SSL/TLS之上，SSL/TLS运行在TCP之上**，所有传输的内容都经过加密的。
- HTTP和HTTPS使用的是完全不同的连接方式，用的**端口**也不一样，前者是**80**，后者是**443**。
- HTTPS可以有效的防止运营商劫持，解决了防劫持的一个大问题。



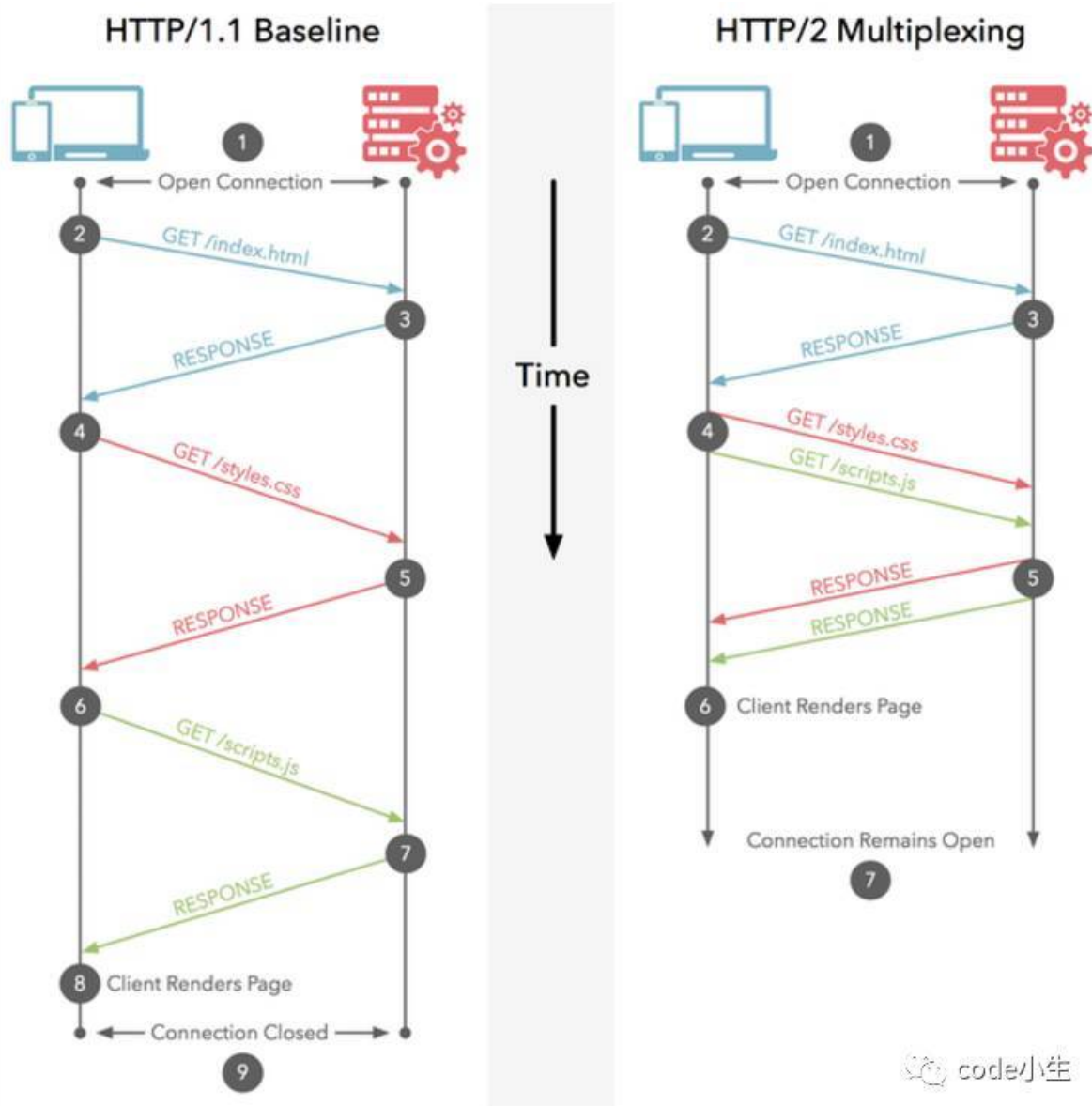


## HTTP2.0和HTTP1.X相比的新特性

- **新的二进制格式 (Binary Format)**，HTTP1.x的解析是基于文本。基于文本协议的格式解析存在天然缺陷，文本的表现形式有多样性，要做到健壮性考虑的场景必然很多，二进制则不同，只认0和1的组合。基于这种考虑HTTP2.0的协议解析决定采用二进制格式，实现方便且健壮。
- **多路复用 (MultiPlexing)**，即连接共享，即每一个request都是用作连接共享机制的。一个request对应一个id，这样一个连接上可以有多个request，每个连接的request可以随机的混杂在一起，接收方可以根据request的 id将request再归属到各自不同的服务端请求里面。
- **header压缩**，如上文中所言，对前面提到过HTTP1.x的header带有大量信息，而且每次都要重复发送，**HTTP2.0使用encoder来减少需要传输的header大小**，通讯双方各自cache一份header fields表，既避免了重复header的传输，又减小了需要传输的大小。
- **服务端推送 (server push)**，同SPDY一样，HTTP2.0也具有server push功能。

## HTTP2.0的多路复用和HTTP1.X中的长连接复用有什么区别？

- **HTTP/1.\*** 一次请求-响应，建立一个连接，用完关闭；每一个请求都要建立一个连接；
- HTTP/1.1 Pipeling解决方式为，若干个请求排队串行化单线程处理，后面的请求等待前面请求的返回才能获得执行机会，一旦有某请求超时等，后续请求只能被阻塞，毫无办法，也就是人们常说的线头阻塞；
- HTTP/2多个请求可同时在一个连接上并行执行。某个请求任务耗时严重，不会影响到其它连接的正常执行；具体如图：



## 2.长连接、短连接的使用场景

http的长连接和短连接实际上是**TCP**的长连接和短连接。

**区分长/短连接：**整个客户端和服务端的通讯过程是利用一个Socket还是多个Socket进行的。

**长连接**适用于**操作频繁/点对点通讯等连接数不太多**的情况，如：一些游戏/即时通讯场景应该使用长连接；如：老师端和学生端的即时通讯教学软件；最经典的案例是：**数据库使用的连接**（如果使用短连接会造成Socket错误）

**短连接**适用于Web【wapWeb/H5等】的http服务，长连接对于服务端来说会耗费一定资源。对于**电子商务Web的访问量可能是千万级别甚至亿万级别的**。如果使用长连接的方式**当一万个用户访问时会占用一万个连接**，假设服务器站点（IIS等）的通信吞吐量只有1千个，那么另外九千人就彻底挂啦，所以并发量大且用户不需要频繁的交互式操作时 用短连接为上策。

### 长连接和短连接的优点和缺点

**长连接**可以省去较多的TCP建立和关闭的操作，减少浪费，节约时间。对于频繁请求资源的客户来说，较适用长连接。Client与server之间的连接如果一直不关闭的话，会存在一个问题，**随着客户端连接越来越多，server早晚有扛不住的时候，**

短连接对于服务器来说管理较为简单，存在的连接都是有用的连接，不需要额外的控制手段。但如果客户请求频繁，将在TCP的建立和关闭操作上浪费时间和带宽。

### 3.如何处理高并发场景下长连接过多带来的问题

## 4.请简单描述http协议的请求报文和响应报文的组成格式

### HTTP请求报文

一个HTTP请求报文由请求行（request line）、请求头部（header）、空行和请求数据4个部分组成，下图给出了请求报文的一般格式。



图 15-4 HTTP 请求报文

#### 1.请求头

请求行由请求方法字段、URL字段和HTTP协议版本字段3个字段组成，它们用空格分隔。例如，GET /index.html HTTP/1.1。

HTTP协议的请求方法有GET、POST、HEAD、PUT、DELETE、OPTIONS、TRACE、CONNECT。

#### 2.请求头部

请求头部由关键字/值对组成，每行一对，关键字和值用英文冒号“:”分隔。请求头部通知服务器有关于客户端请求的信息，典型的请求头有：

User-Agent：产生请求的浏览器类型。

Accept：客户端可识别的内容类型列表。

Host：请求的主机名，允许多个域名同处一个IP地址，即虚拟主机。

#### 3.空行

最后一个请求头之后是一个空行，发送回车符和换行符，通知服务器以下不再有请求头。

## 4.请求数据

请求数据不在GET方法中使用，而是在POST方法中使用。POST方法适用于需要客户填写表单的场合。与请求数据相关的最常使用的请求头是Content-Type和Content-Length。

## HTTP响应报文



HTTP响应也由三个部分组成，分别是：状态行、消息报头、响应正文。

其中，HTTP-Version表示服务器HTTP协议的版本；Status-Code表示服务器发回的响应状态代码；Reason-Phrase表示状态代码的文本描述。状态代码由三位数字组成，第一个数字定义了响应的类别，且有五种可能取值。

- 1xx：指示信息--表示请求已接收，继续处理。
- 2xx：成功--表示请求已被成功接收、理解、接受。
- 3xx：重定向--要完成请求必须进行更进一步的操作。
- 4xx：客户端错误--请求有语法错误或请求无法实现。
- 5xx：服务器端错误--服务器未能实现合法的请求。

常见状态代码、状态描述的说明如下。

- 200 OK：客户端请求成功。
- 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。
- 403 Forbidden：服务器收到请求，但是拒绝提供服务。
- 404 Not Found：请求资源不存在，举个例子：输入了错误的URL。
- 500 Internal Server Error：服务器发生不可预期的错误。
- 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，举个例子：HTTP/1.1 200 OK (CRLF) 。

## 5.http 之session和cookie

- 由于HTTP协议是无状态的协议，所以服务端需要记录用户的状态时，就需要用某种机制来识具体的用户，这个机制就是Session.典型的场景比如购物车，当你点击下单按钮时，由于HTTP协议无状态，所以并不知道是哪个用户操作的，所以服务端要为特定的用户创建了特定的Session，用于标识这个用户，并且跟踪用户，这样才知道购物车里面有几本书。这个Session是保存在服务端的，有一个唯一标识。在服务端保存Session的方法很多，内存、数据库、文件都有。集群的时候也要考虑Session的转移，在大型的网站，一般会有专门的Session服务器集群，用来保存用户会话，这个时候 Session 信息都是放在内存的，使用一些缓存服务比如Memcached之类的来放Session。
1. 思考一下服务端如何识别特定的客户？这个时候Cookie就登场了。每次HTTP请求的时候，客户端都会发送相应的Cookie信息到服务端。实际上大多数的应用都是用 Cookie 来实现Session跟踪的，第一次创建Session的时候，服务端会在HTTP协议中告诉客户端，需要在 Cookie 里面记录一个Session ID，以后每次请求把这个会话ID发送到服务器，我就知道你是谁了。有人问，如果客户端的浏览器禁用了 Cookie 怎么办？一般这种情况下，会使用一种叫做URL重写的技术来进行会话跟踪，即每次HTTP交互，URL后面都会被附加上一个诸如 sid=xxxxx 这样的参数，服务端据此来识别用户。
  2. Cookie其实还可以用在一些方便用户的场景下，设想你某次登陆过一个网站，下次登录的时候不想再次输入账号了，怎么办？这个信息可以写到Cookie里面，访问网站的时候，网站页面的脚本可以读取这个信息，就自动帮你把用户名给填了，能够方便一下用户。这也是Cookie名称的由来，给用户的一点甜头。所以，总结一下：Session是在服务端保存的一个数据结构，用来跟踪用户的状态，这个数据可以保存在集群、数据库、文件中；Cookie是客户端保存用户信息的一种机制，用来记录用户的一些信息，也是实现Session的一种方式

## 6.socket

### 3、SOCKET原理

#### 3.1套接字 (socket) 概念

套接字 (socket) 是通信的基石，是支持TCP/IP协议的网络通信的基本操作单元。它是网络通信过程中端点的抽象表示，包含进行网络通信必须的五种信息：连接使用的协议，本地主机的IP地址，本地进程的协议端口，远地主机的IP地址，远地进程的协议端口。

应用层通过传输层进行数据通信时，TCP会遇到同时为多个应用程序进程提供并发服务的问题。多个TCP连接或多个应用程序进程可能需要通过同一个TCP协议端口传输数据。为了区别不同的应用程序进程和连接，许多计算机操作系统为应用程序与TCP / IP协议交互提供了套接字(Socket)接口。应用层可以和传输层通过Socket接口，区分来自不同应用程序进程或网络连接的通信，实现数据传输的并发服务。

#### 3.2 建立socket连接

建立Socket连接至少需要一对套接字，其中一个运行于客户端，称为ClientSocket，另一个运行于服务器端，称为ServerSocket。

套接字之间的连接过程分为三个步骤：服务器监听，客户端请求，连接确认。

**服务器监听：**服务器端套接字并不定位具体的客户端套接字，而是处于等待连接的状态，实时监控网络状态，等待客户端的连接请求。

**客户端请求：**指客户端的套接字提出连接请求，要连接的目标是服务器端的套接字。为此，客户端的套接字必须首先描述它要连接的服务器的套接字，指出服务器端套接字的地址和端口号，然后就向服务器端套接字提出连接请求。

**连接确认：**当服务器端套接字监听到或者说接收到客户端套接字的连接请求时，就响应客户端套接字的请求，建立一个新的线程，把服务器端套接字的描述发给客户端，一旦客户端确认了此描述，双方就正式建立连接。而服务器端套接字继续处于监听状态，继续接收其他客户端套接字的连接请求。

#### 4、SOCKET连接与TCP/IP连接

创建Socket连接时，可以指定使用的传输层协议，Socket可以支持不同的传输层协议（TCP或UDP），当使用TCP协议进行连接时，该Socket连接就是一个TCP连接。

socket则是对TCP/IP协议的封装和应用（程序员层面上）。也可以说，TCP/IP协议是传输层协议，主要解决数据如何在网络中传输，而HTTP是应用层协议，主要解决如何包装数据。关于TCP/IP和HTTP协议的关系，网络有一段比较容易理解的介绍：

“我们在传输数据时，可以只使用（传输层）TCP/IP协议，但是那样的话，如果没有应用层，便无法识别数据内容，如果想要使传输的数据有意义，则必须使用到应用层协议，应用层协议有很多，比如HTTP、FTP、TELNET等，也可以自己定义应用层协议。WEB使用HTTP协议作应用层协议，以封装HTTP文本信息，然后使用TCP/IP做传输层协议将它发到网络上。”

我们平时说的最多的socket是什么呢，实际上socket是对TCP/IP协议的封装，Socket本身并不是协议，而是一个调用接口（API），通过Socket，我们才能使用TCP/IP协议。实际上，Socket跟TCP/IP协议没有必然的联系。Socket编程接口在设计的时候，就希望也能适应其他的网络协议。所以说，Socket的出现只是使得程序员更方便地使用TCP/IP协议栈而已，是对TCP/IP协议的抽象，从而形成了我们知道的一些最基本的函数接口，比如create、listen、connect、accept、send、read和write等等。网络有一段关于socket和TCP/IP协议关系的说法比较容易理解：

“TCP/IP只是一个协议栈，就像操作系统的运行机制一样，必须要具体实现，同时还要提供对外的操作接口。这个就像操作系统会提供标准的编程接口，比如win32编程接口一样，TCP/IP也要提供可供程序员做网络开发所用的接口，这就是Socket编程接口。”

实际上，传输层的TCP是基于网络层的IP协议的，而应用层的HTTP协议又是基于传输层的TCP协议的，而Socket本身不算是协议，就像上面所说，它只是提供了一个针对TCP或者UDP编程的接口。socket是对端口通信开发的工具，它要更底层一些。

#### 5、Socket连接与HTTP连接

由于通常情况下Socket连接就是TCP连接，因此Socket连接一旦建立，通信双方即可开始相互发送数据内容，直到双方连接断开。但在实际网络应用中，客户端到服务器之间的通信往往需要穿越多个中间节点，例如路由器、网关、防火墙等，大部分防火墙默认会关闭长时间处于非活跃状态的连接而导致Socket连接断开，因此需要通过轮询告诉网络，该连接处于活跃状态。

而HTTP连接使用的是“请求—响应”的方式，不仅在请求时需要先建立连接，而且需要客户端向服务器发出请求后，服务器端才能回复数据。很多情况下，需要服务器端主动向客户端推送数据，保持客户端与服务器数据的实时与同步。此时若双方建立的是Socket连接，服务器就可以直接将数据传送给客户端；若双方建立的是HTTP连接，则服务器需要等到客户端发送一次请求后才能将数据传回给客户端，因此，客户端定时向服务器端发送连接请求，不仅可以保持在线，同时也是在“询问”服务器是否有新的数据，如果有就将数据传给客户端。

http协议是应用层的协议

有个比较形象的描述：HTTP是轿车，提供了封装或者显示数据的具体形式；Socket是发动机，提供了网络通信的能力。

两个计算机之间的交流无非是两个端口之间的数据通信,具体的数据会以什么样的形式展现是以不同的应用层协议来定义的如HTTP FTP..

## 7.http缓存

---

[彻底弄懂HTTP缓存机制及原理](#)

强制缓存、对比缓存 对于强制缓存，服务器通知浏览器一个缓存时间，在缓存时间内，下次请求，直接用缓存，不在时间内，执行比较缓存策略。对于比较缓存，将缓存信息中的Etag和Last-Modified通过请求发送给服务器，由服务器校验，返回304状态码时，浏览器直接使用缓存。

## 场景题

---

[10亿个数中找出最大的10000个数之top K问题](#)

---