# **一：并发编程**

## 1：同步(synchronized)

## 2：volatile

作用：使变量在多个线程间可见。

只具备可见性，不具备原子性：

volatile和synchronized，性能要比synchronized强很多，，不能替代synchronized的同步功能

每一个线程都有一块工作内存区，存放着所有线程共享的主内存中的变量的值的拷贝

volatile的作用是强制线程执行引擎到主内存去读取变量，而不去线程工作内存区中去读取，变量在多个线程间可见

## 3:多线程通信(wait notify/notifyAll)

撸一段：wait notify

countDownLatch、

## 4：threadLocal：线程局部变量，与锁无关的线程安全解决方案，减少锁的竞争

## 5：多线程和单例

单例模式：

饥饿模式

懒汉模式

多线程的单例实现

1：dubble check instance

2: inner static

## 

## 6同步类容器&并发类容器

同步类容器：vector,hashtable等底层机制使用synchronized实现，牺牲性能,古典

并发类容器：ConcurrentHashMap代替散列

CopyOnWriteArrsyList 代替vector

CopyOnWriteArraySet

并发的queue,ConcurrentLinkedQueue,LinkedBlockingQueue

ArrayBlockingQueue,PriorityBlockingQueue,SynchronoutQueue

## 7:concurrentHashMap

concurrentHashMap内部使用段（segment）来表示不同的部分，每个段其实就是一个小的hashtable，有自己的锁，只要多个修改操作发生在不同的段上

它们就可以并发的执行，把一个整体部分分分成了16个段，也就是最高支持16个线程的并发的修改操作，多线程场景时减小锁的粒度从而降低锁竞争的一种方案。

## 8：copyonwrite

使用场景：读多写少的时候使用

copyonwrite容器即为写时复制的容器，当我们往一个容器添加元素的时候，不直接往当前容器添加，而是先将当前容器进行copy,复制一个新的容器，然后往新的容器

进行添加元素，添加完成之后，再将圆融期的引用指向新的容器。好处的，对copyonwrite进行并发的读的时候，不需要加锁。利用了读写分离的思想。

## 9：并发queue

一个concurrentLinkedQueue为代表的高性能queue

一个blockingQueue接口为代表的阻塞队列

blockingQueue:ArrayBlockingQueue

LinkedBlockingQueue

PriorityBlockingQueue

DelayQueue

## 10并行设计模式（future,master-worker,生产者消费者）

并行设计模式属于设计优化的一部分，对一些常用的多线程结构的总结和抽象

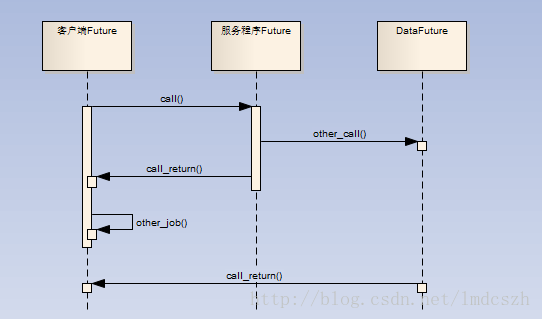
模式：future ,master-worker和生产者消费者模型

future:

该模型是将异步请求和代理模式联合的模型产物

客户端发送一个长时间的请求，服务端不需要等待该数据处理完成就立即返回一个伪造

的代理数据，用户也无需要等待，先去执行其他的若干操作，再去调用服务器已经完成组装的真实数据。该模型充分利用了等待的时间片段



future模式的核心架构

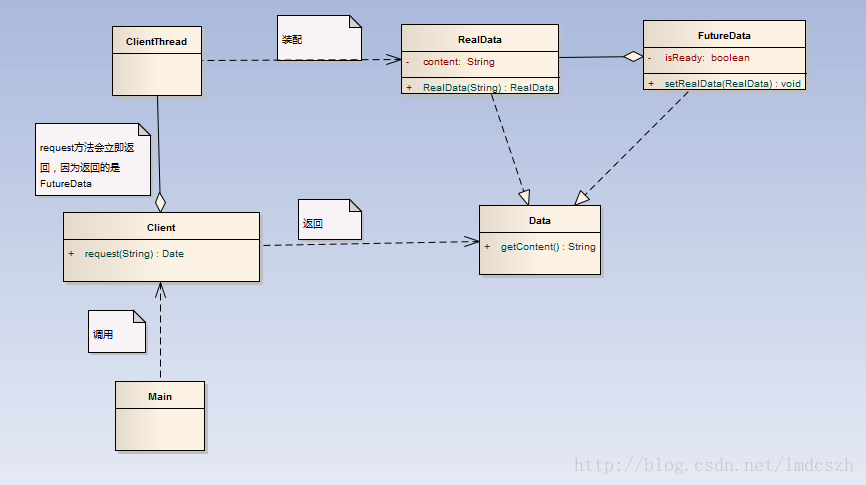
Main:启动系统，调用client发出请求

client返回data对象，理解返回futuredata,并开启，client线程装配realdata

Data:返回数据的接口

Futuredata:future数据，构造很快，这是一个虚拟的数据，需要装配realdata

Realdata:真实的数据，构造比较慢



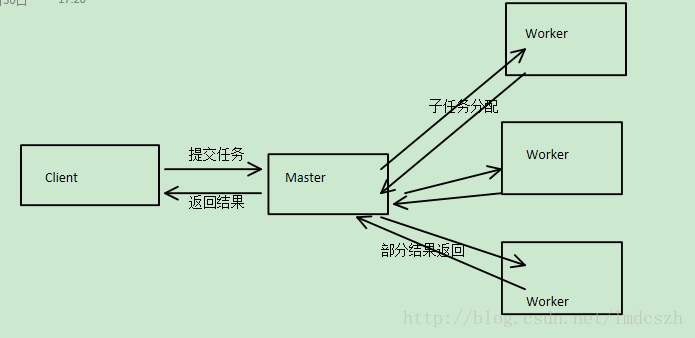
master-worker

常用的并行计算的模式

核心思想是系统由两类进程协作工作：master进程和worker进程。master负责接收和分 配任务,worker负责处理子任务。

各个worker子进程处理完成后，会将结果返回给master，由master做归纳总结。好处是能将一个大任务，分解成若干个小任务，并行执行，从而提高系统的吞吐量。

生产者消费者模型



## 生产者消费者

## 11线程池

多任务的执行框架executor

为了更好的控制多线程，JDK提供了一套线程框架Executor，帮助开发人员有效的进行线程控制，在java.util.concurrent包中，是JDK并发包的核心。期中有一个比较重要的类：Executors，线程工厂的核心

Executors创建线程池方法

New fixedThreadPool,返回一个固定数量的线程池，当有一个任务提交时候，若线程池中空闲，则

立即执行，若没有，则会被暂缓在一个任务队列中等待有空闲的线程去执行

New singleThreadExecutor，创建一个线程的线程池

New cachedThreadPool，返回一个可根据实际情况调整线程个数的线程池，每一个空线程会在60

秒回收

New scheduledThreadPool

## 12java.util.concurrent工具类

## cyclicBarrier

使用场景：运动员都准备好了，才一起去跑

## countDownLatch

### 

经常用于坚持某个初始化操作，等初始化完成以后，通知主线程继续工作

countDown = New countDownLatch(2);

countDown.countDown();

countDown.await();

**注：countDownLatch:一个线程等待，其他线程countDown**

**cyclicBarrier：多个线程等待**

## callable和future

Runnable和Callable的区别是，  
(1)Callable规定的方法是call(),Runnable规定的方法是run().  
(2)Callable的任务执行后可返回值，而Runnable的任务是不能返回值得  
(3)call方法可以抛出异常，run方法不可以

1. 运行Callable任务可以拿到一个Future对象，Future 表示异步计算的结果

## 13重入锁，读写锁

嗅探锁定，多路分支的功能

reentrantLock:重入锁

reentrantReadWriteLock，实现读写分离，读读共享，写写斥斥，读写互斥

## 14并发执行框架（disruptor(生产消费的模型)）

事件源驱动方式，在一个线程里面每秒处理600万订单

第一：建立一个event类

第二：建立一个工厂event类，用于创建event类实例对象

第三：需要有一个监听事件类，用于处理数据（event类）

第四：测试代码编写，实例化disruptor实例，配置一系列参宿，对disruptor实例绑定监听事件类。

第五：在disruptor中

# 二：网络编程

套接字连接过程：四个步骤，服务器监听，客户端请求服务器，服务器确认，客户端确认。

## 1:传统的BIO：blocking IO

缺点：客户端增大的时候，服务器支撑不起太多的客户端连接

### jdk1.5以前采用伪异步的方式进行解决

方法：在服务端采用线程池进行处理

## 2：异步 非阻塞 jdk1.7以后的版本（NIO->AIO）

BIO（同步阻塞）->NIO（同步非阻塞）->AIO（异步非阻塞）

IO(BIO)和NIO的区别

本质是阻塞和非阻塞的区别

非阻塞：应用程序直接获取已经准备好的数据，无需等待

NIO（JDK1.5以后）

AIO（JDK1.7以后）

## 3：Netty

### 3.1连接步骤

1：服务器端TCP内核模块维护两个队列，我们称为A，B

2：客户端向服务器端connect的时候，会发送带有SYN标志的包（第一次握手）

3：服务器收到客户端发来的SYN时，向客户端发送SYN+ACK确认（第二次握手）

4：此时TCP内核模块把客户端连接加入到A队列中，然后服务器收到客户端发来的ACK时（第 三此握手）

5：TCP内核模块把客户端从A队列移到B队列中，连接完成，应用程序accept会返回

即accept从B队列中取出完成三次握手的连接

6：A队列和B队列的长度之和是backblog，当A，B队列的长度之和大于Backblog时候，新连接会被TCP内核拒绝掉

支持绑定多个端口，走一个处理器，对每一个客户端做出不同的响应

netty独立搞一个jar包，放在服务器上，tomcat独立部署

### 3.2TCP粘包拆包

问题描述

本质：TCP基于流的传输并不是一个数据包队列，而是一个字节队列。

解决方案

本质：发送端给接收端一个标记，告诉服务端结束标记是什么

方法1：消息定长，每个报文固定大小

方法2：包尾增加特殊符号，比如回车换行

方法3：消息分为消息头和消息尾，消息头中包含消息长度的字段，

### 3.3netty实现文件的上传和下载