多线程设计模式详解

### 1、[single threaded execution](http://pan-java.iteye.com/blog/421610)

调用线程

**Java代码  [收藏代码](javascript:void())**

1. **public** **class** UserThread **extends** Thread {
2. **private** **final** Gate gate;
3. **private** **final** String myname;
4. **private** **final** String myaddress;
5. **public** UserThread(Gate gate, String myname, String myaddress) {
6. **this**.gate = gate;
7. **this**.myname = myname;
8. **this**.myaddress = myaddress;
9. }
10. **public** **void** run() {
11. System.out.println(myname + " BEGIN");
12. **while** (**true**) {
13. gate.pass(myname, myaddress);
14. }
15. }
16. }

同时只允许一个对象访问锁块.(类似于synchronized 标示的块)   
这样做的主要原因是因为保护类变量不同时被多个线程修改   
方法内部的变量不会在多线程同时修改.   
  
任务

**Java代码  [收藏代码](javascript:void())**

1. **public** **class** Gate {
2. **private** **int** counter = 0;
3. **private** String name = "Nobody";
4. **private** String address = "Nowhere";
5. //jdk1.5新特性 效率高于synchronized
6. **private** Lock passLock = **new** ReentrantLock();
8. **private** Lock toStringLock = **new** ReentrantLock();
10. **public** **void** pass(String name, String address) {
11. //使用锁机制
12. passLock.lock();
13. **try** {
14. **this**.counter++;
15. **this**.name = name;
16. **this**.address = address;
17. check();
18. } **finally** {
19. //解锁
20. passLock.unlock();
21. }
22. }
24. **public** String toString() {
25. toStringLock.lock();
26. **try** {
27. **return** "No." + counter + ": " + name + ", " + address;
28. } **finally** {
29. toStringLock.unlock();
30. }
31. }
33. **private** **void** check() {
34. **if** (name.charAt(0) != address.charAt(0)) {
35. System.out.println("\*\*\*\*\* BROKEN \*\*\*\*\* " + toString());
36. }
37. }
38. }

主程序

**Java代码  [收藏代码](javascript:void())**

1. **public** **class** Main {
2. **public** **static** **void** main(String[] args) {
3. System.out.println("Testing Gate, hit CTRL+C to exit.");
4. Gate gate = **new** Gate();
5. **new** UserThread(gate, "Alice", "Alaska").start();
6. **new** UserThread(gate, "Bobby", "Brazil").start();
7. **new** UserThread(gate, "Chris", "Canada").start();
8. }
9. }

### 2、[immutable](http://pan-java.iteye.com/blog/421951)

对象

**Java代码  [收藏代码](javascript:void())**

1. //类定义为 final
2. **public** **final** **class** Person {
3. //类变量定义为 final
4. **private** **final** String name;
5. **private** **final** String address;
7. **public** Person(String name, String address) {
8. **this**.name = name;
9. **this**.address = address;
10. }
12. **public** String getName() {
13. **return** name;
14. }
16. **public** String getAddress() {
17. **return** address;
18. }
20. //name ,address  是不能修改的,所在多线程下安全的
21. **public** String toString() {
22. **return** "[ Person: name = " + name + ", address = " + address + " ]";
23. }

线程

**Java代码  [收藏代码](javascript:void())**

1. **public** **class** PrintPersonThread **extends** Thread {
2. **private** Person person;
4. **public** PrintPersonThread(Person person) {
5. **this**.person = person;
6. }
8. **public** **void** run() {
9. **while** (**true**) {
10. //将调用Person 的toString 方法
11. //当多线程情况Person的name,address是final的 不能被修改所以是线程安全
12. System.out.println(Thread.currentThread().getName() + " prints "
13. + person);
14. }
15. }
16. }

**Java代码  [收藏代码](javascript:void())**

1. **public** **class** Main {
2. **public** **static** **void** main(String[] args) {
3. Person alice = **new** Person("Alice", "Alaska");
4. **new** PrintPersonThread(alice).start();
5. **new** PrintPersonThread(alice ).start();
6. **new** PrintPersonThread(alice ).start();
7. }
8. }

用final标示比使用锁块和syschronized效率要好   
同时 Integer ,等一些基本类型的包装类 都是使用immutable 模式

### 3、[suspension](http://pan-java.iteye.com/blog/422387)

请求对象

**Java代码  [收藏代码](javascript:void())**

1. **public** **class** Request {
2. **private** **final** String name;
3. **public** Request(String name) {
4. **this**.name = name;
5. }
6. **public** String getName() {
7. **return** name;
8. }
9. **public** String toString() {
10. **return** "[ Request " + name + " ]";
11. }
12. }

工作缓存器

**Java代码  [收藏代码](javascript:void())**

1. **public** **class** RequestQueue {
2. //采用LinkedList 保存客户请求
3. **private** **final** LinkedList queue = **new** LinkedList();
4. //服务器获取请求
5. //保证同时只一个服务器执行这个方法
6. **public** **synchronized** Request getRequest() {
7. //当客户请求队列为空时等待
8. **while** (queue.size() <= 0) {
9. **try** {
10. wait();//等到客户添加请求
11. } **catch** (InterruptedException e) {
12. }
13. }
14. **return** (Request)queue.removeFirst();
15. }
17. //客户添加请求
18. //保证同时只一个客户执行这个方法
19. **public** **synchronized** **void** putRequest(Request request) {
20. queue.addLast(request);
21. notifyAll();//唤醒wait()方法,执行程序
22. }
23. }

工作缓存器jdk1.5新特性实现

**Java代码  [收藏代码](javascript:void())**

1. **public** **class** RequestQueue {
2. **private** **final** LinkedList queue = **new** LinkedList();
4. **private** Lock lock = **new** ReentrantLock();
6. **private** Condition serverCondition = lock.newCondition();
8. **private** Condition clientCondition = lock.newCondition();
10. **public** Request getRequest() {
11. lock.lock();
12. **try** {
13. **while** (queue.size() <= 0) {
14. //类似wait();
15. serverCondition.await();
16. }
17. } **catch** (InterruptedException e) {
18. } **finally** {
19. lock.unlock();
20. }
22. **return** (Request) queue.removeFirst();
23. }
25. **public** **void** putRequest(Request request) {
26. lock.lock();
27. **try** {
28. queue.addLast(request);
29. //类似notifyAll();
30. clientCondition .signalAll();
32. } **finally** {
33. lock.unlock();
34. }
35. }
36. }

工作缓存器jdk1.5新特性实现

**Java代码  [收藏代码](javascript:void())**

1. **public** **class** NewRequestQueue {
2. //一个特殊的Queue下面有详细的说明
3. **private** **final** **static** BlockingQueue<Request> basket = **new** ArrayBlockingQueue<Request>(
4. 10);;
6. **public** Request getRequest() {
8. **try** {
9. **return** basket.take();
10. } **catch** (Exception e) {
11. **throw** **new** RuntimeException(e);
12. }
13. }
15. **public** **void** putRequest(Request request) {
17. **try** {
18. basket.put(request);
19. } **catch** (Exception e) {
20. **throw** **new** RuntimeException(e);
21. }
22. }
24. }

BlockingQueue 是一种特殊的Queue，若BlockingQueue 是空的，从   
BlockingQueue 取东西的操作将会被阻断进入等待状态直到BlocingkQueue 进   
了新货才会被唤醒。同样，如果BlockingQueue 是满的任何试图往里存东西的   
操作也会被阻断进入等待状态，直到BlockingQueue 里有新的空间才会被唤醒   
继续操作。BlockingQueue 提供的方法主要有：   
add(anObject): 把anObject 加到BlockingQueue 里，如果   
BlockingQueue 可以容纳返回true，否则抛出IllegalStateException 异常。   
offer(anObject)：把anObject 加到BlockingQueue 里，如果   
BlockingQueue 可以容纳返回true，否则返回false。   
put(anObject)：把anObject 加到BlockingQueue 里，如果BlockingQueue   
没有空间，调用此方法的线程被阻断直到BlockingQueue 里有新的空间再继   
续。   
poll(time)：取出BlockingQueue 里排在首位的对象，若不能立即取出可   
等time 参数规定的时间。取不到时返回null。   
take()：取出BlockingQueue 里排在首位的对象，若BlockingQueue 为   
空，阻断进入等待状态直到BlockingQueue 有新的对象被加入为止。   
根据不同的需要 BlockingQueue 有4 种具体实现：   
ArrayBlockingQueue：规定大小的BlockingQueue，其构造函数必须带   
一个int 参数来指明其大小。其所含的对象是以FIFO（先入先出）顺序排序   
的。   
LinkedBlockingQueue：大小不定的BlockingQueue，若其构造函数带一   
个规定大小的参数，生成的BlockingQueue 有大小限制，若不带大小参数，   
所生成的BlockingQueue 的大小由Integer.MAX\_VALUE 来决定。其所含的   
对象是以FIFO（先入先出）顺序排序的。LinkedBlockingQueue 和   
ArrayBlockingQueue 比较起来，它们背后所用的数据结构不一样，导致   
LinkedBlockingQueue 的数据吞吐量要大于ArrayBlockingQueue，但在线程   
数量很大时其性能的可预见性低于ArrayBlockingQueue。   
PriorityBlockingQueue：类似于LinkedBlockingQueue，但其所含对象的   
排序不是FIFO，而是依据对象的自然排序顺序或者是构造函数所带的   
Comparator 决定的顺序。   
SynchronousQueue：特殊的BlockingQueue，对其的操作必须是放和取   
交替完成的。   
  
  
客户线程

**Java代码  [收藏代码](javascript:void())**

1. **public** **class** ClientThread **extends** Thread {
2. **private** Random random;
3. **private** RequestQueue requestQueue;
4. **public** ClientThread(RequestQueue requestQueue, String name, **long** seed) {
5. **super**(name);
6. **this**.requestQueue = requestQueue;
7. **this**.random = **new** Random(seed);
8. }
9. **public** **void** run() {
10. **for** (**int** i = 0; i < 10000; i++) {
11. Request request = **new** Request("No." + i);
12. System.out.println(Thread.currentThread().getName() + " requests " + request);
13. //放入请求
14. requestQueue.putRequest(request);
15. **try** {
16. Thread.sleep(random.nextInt(1000));
17. } **catch** (InterruptedException e) {
18. }
19. }
20. }
21. }

服务器线程

**Java代码  [收藏代码](javascript:void())**

1. **public** **class** ServerThread **extends** Thread {
2. **private** Random random;
3. **private** RequestQueue requestQueue;
4. **public** ServerThread(RequestQueue requestQueue, String name, **long** seed) {
5. **super**(name);
6. **this**.requestQueue = requestQueue;
7. **this**.random = **new** Random(seed);
8. }
9. **public** **void** run() {
10. **for** (**int** i = 0; i < 10000; i++) {
11. Request request = requestQueue.getRequest();
12. System.out.println(Thread.currentThread().getName() + " handles  " + request);
13. **try** {
14. Thread.sleep(random.nextInt(1000));
15. } **catch** (InterruptedException e) {
16. }
17. }
18. }
19. }

测试类

**Java代码  [收藏代码](javascript:void())**

1. **public** **class** Main {
2. **public** **static** **void** main(String[] args) {
3. RequestQueue requestQueue = **new** RequestQueue();
4. **new** ClientThread(requestQueue, "Alice", 3141592L).start();
5. **new** ServerThread(requestQueue, "Bobby", 6535897L).start();
6. }
7. }

4、Balking

//操作数据

**Java代码  [收藏代码](javascript:void())**

1. **public** **class** Data {
2. **private** String filename;    //修改是的名字
3. **private** String content;     // 资料的内容
4. **private** **boolean** changed;    //修改后的内容还没存储的话，值为true
6. **public** Data(String filename, String content) {
7. **this**.filename = filename;
8. **this**.content = content;
9. **this**.changed = **true**;
10. }
12. // 修改资料内容
13. **public** **synchronized** **void** change(String newContent) {
14. content = newContent;
15. changed = **true**;
16. }
18. // 若有资料修改，就存储到挡安里
19. **public** **synchronized** **void** save() **throws** IOException {
20. //如果内容没有改变就返回
21. **if** (!changed) {
22. System.out.println(Thread.currentThread().getName() + " balks");
23. **return**;
24. }
25. doSave();
26. changed = **false**;
27. }
29. // 实际资料储存到挡案里用的方法
30. **private** **void** doSave() **throws** IOException {
31. System.out.println(Thread.currentThread().getName() + " calls doSave, content = " + content);
32. Writer writer = **new** FileWriter(filename);
33. writer.write(content);
34. writer.close();
35. }
36. }

保存线程

**Java代码  [收藏代码](javascript:void())**

1. **public** **class** SaverThread **extends** Thread {
2. **private** Data data;
3. **public** SaverThread(String name, Data data) {
4. **super**(name);
5. **this**.data = data;
6. }
7. **public** **void** run() {
8. **try** {
9. **while** (**true**) {
10. //每1秒保存一次
11. data.save();            // 存储资料
12. Thread.sleep(1000);     // 休息约1秒
13. }
14. } **catch** (IOException e) {
15. e.printStackTrace();
16. } **catch** (InterruptedException e) {
17. e.printStackTrace();
18. }
19. }
20. }

改变线程

**Java代码  [收藏代码](javascript:void())**

1. **public** **class** ChangerThread **extends** Thread {
2. **private** Data data;
3. **private** Random random = **new** Random();
4. **public** ChangerThread(String name, Data data) {
5. **super**(name);
6. **this**.data = data;
7. }
8. **public** **void** run() {
9. **try** {
10. **for** (**int** i = 0; **true**; i++) {
11. data.change("No." + i);             //修改资料
12. Thread.sleep(random.nextInt(1000)); // 模拟去做别的事情
13. data.save();                        // 明确的要求存档
14. }
15. } **catch** (IOException e) {
16. e.printStackTrace();
17. } **catch** (InterruptedException e) {
18. e.printStackTrace();
19. }
20. }
21. }

测试线程

**Java代码  [收藏代码](javascript:void())**

1. **public** **class** Main {
2. **public** **static** **void** main(String[] args) {
3. Data data = **new** Data("data.txt", "(empty)");
4. **new** ChangerThread("ChangerThread", data).start();
5. **new** SaverThread("SaverThread", data).start();
6. }
7. }

适用性   
1.不需要刻意去执行的时候   
2.不想等待警戒条件成立时

### 5、[ProducerConsumer](http://pan-java.iteye.com/blog/423060)

放蛋糕的桌子

**Java代码  [收藏代码](javascript:void())**

1. **public** **class** Table {
2. **private** **final** String[] buffer;
3. **private** **int** tail;  /下一个放put的地方
4. **private** **int** head;  //下一个放的take地方
5. **private** **int** count; // buffer内的蛋糕数
6. **public** Table(**int** count) {
7. **this**.buffer = **new** String[count];
8. **this**.head = 0;
9. **this**.tail = 0;
10. **this**.count = 0;
11. }
12. // 放置蛋糕
13. **public** **synchronized** **void** put(String cake) **throws** InterruptedException {
14. System.out.println(Thread.currentThread().getName() + " puts " + cake);
15. **while** (count >= buffer.length) {
16. wait();//当蛋糕数大于容器大小是等待
17. }
18. buffer[tail] = cake;
19. tail = (tail + 1) % buffer.length;
20. count++;
21. notifyAll();//唤醒等待线程
22. }

25. // 取得蛋糕
26. **public** **synchronized** String take() **throws** InterruptedException {
27. **while** (count <= 0) {
28. wait();//当蛋糕数为0时,等待
29. }
30. String cake = buffer[head];
31. head = (head + 1) % buffer.length;
32. count--;
33. notifyAll();//唤醒等待线程
34. System.out.println(Thread.currentThread().getName() + " takes " + cake);
35. **return** cake;
36. }
37. }

吃线程

**Java代码  [收藏代码](javascript:void())**

1. **public** **class** EaterThread **extends** Thread {
2. **private** **final** Random random;
3. **private** **final** Table table;
4. **public** EaterThread(String name, Table table, **long** seed) {
5. **super**(name);
6. **this**.table = table;
7. **this**.random = **new** Random(seed);
8. }
9. **public** **void** run() {
10. **try** {
11. **while** (**true**) {
12. String cake = table.take();
13. Thread.sleep(random.nextInt(1000));
14. }
15. } **catch** (InterruptedException e) {
16. }
17. }
18. }

做线程

**Java代码  [收藏代码](javascript:void())**

1. **public** **class** MakerThread **extends** Thread {
2. **private** **final** Random random;
3. **private** **final** Table table;
4. **private** **static** **int** id = 0; //  蛋糕的流水号(所有厨师共通)
5. **public** MakerThread(String name, Table table, **long** seed) {
6. **super**(name);
7. **this**.table = table;
8. **this**.random = **new** Random(seed);
9. }
10. **public** **void** run() {
11. **try** {
12. **while** (**true**) {
13. Thread.sleep(random.nextInt(1000));
14. String cake = "[ Cake No." + nextId() + " by " + getName() + " ]";
15. table.put(cake);
16. }
17. } **catch** (InterruptedException e) {
18. }
19. }
20. **private** **static** **synchronized** **int** nextId() {
21. **return** id++;
22. }
23. }

测试线程

**Java代码  [收藏代码](javascript:void())**

1. **public** **class** Main {
2. **public** **static** **void** main(String[] args) {
3. Table table = **new** Table(3);     // 建立可以放置3個蛋糕的桌子
4. **new** MakerThread("MakerThread-1", table, 31415).start();
5. **new** MakerThread("MakerThread-2", table, 92653).start();
6. **new** MakerThread("MakerThread-3", table, 58979).start();
7. **new** EaterThread("EaterThread-1", table, 32384).start();
8. **new** EaterThread("EaterThread-2", table, 62643).start();
9. **new** EaterThread("EaterThread-3", table, 38327).start();
10. }
11. }

### 6、[Read-Write Lock Pattern](http://pan-java.iteye.com/blog/423137)

jdk1.5新特性读写锁

**Java代码  [收藏代码](javascript:void())**

1. **public** **class** Data {
2. **private** **final** **char**[] buffer;
3. //private final ReadWriteLock lock = new ReadWriteLock();
4. //读写锁
5. **private** ReadWriteLock lock = **new** ReentrantReadWriteLock();
7. //生成读锁
8. **private** Lock readLock = lock.readLock();
9. //生成写锁
10. **private** Lock writeLock = lock.writeLock();
12. **public** Data(**int** size) {
13. **this**.buffer = **new** **char**[size];
14. **for** (**int** i = 0; i < buffer.length; i++) {
15. buffer[i] = '\*';
16. }
17. }
19. **public** **char**[] read() **throws** InterruptedException {
20. readLock.lock();
21. **try** {
22. **return** doRead();
23. } **finally** {
24. readLock.unlock();
25. }
26. }
28. **public** **void** write(**char** c) **throws** InterruptedException {
29. writeLock.lock();
30. **try** {
31. doWrite(c);
32. } **finally** {
33. writeLock.unlock();
34. }
35. }
37. **private** **char**[] doRead() {
38. **char**[] newbuf = **new** **char**[buffer.length];
39. **for** (**int** i = 0; i < buffer.length; i++) {
40. newbuf[i] = buffer[i];
41. }
42. slowly();
43. **return** newbuf;
44. }
46. **private** **void** doWrite(**char** c) {
47. **for** (**int** i = 0; i < buffer.length; i++) {
48. buffer[i] = c;
49. slowly();
50. }
51. }
53. **private** **void** slowly() {
54. **try** {
55. Thread.sleep(50);
56. } **catch** (InterruptedException e) {
57. }
58. }
59. }

### 7、[Thread-Per-Message](http://pan-java.iteye.com/blog/423169)

代理执行类

**Java代码  [收藏代码](javascript:void())**

1. **public** **class** Host {
2. **private** **final** Helper helper = **new** Helper();
3. **public** **void** request(**final** **int** count, **final** **char** c) {
4. System.out.println("    request(" + count + ", " + c + ") BEGIN");
5. //让执行任务起动一个新线程去执行,不等待方法返回
6. **new** Thread() {
7. **public** **void** run() {
8. helper.handle(count, c);
9. }
10. }.start();
11. System.out.println("    request(" + count + ", " + c + ") END");
12. }
13. }

执行类

**Java代码  [收藏代码](javascript:void())**

1. **public** **class** Helper {
2. **public** **void** handle(**int** count, **char** c) {
3. System.out.println("        handle(" + count + ", " + c + ") BEGIN");
4. **for** (**int** i = 0; i < count; i++) {
5. slowly();
6. System.out.print(c);
7. }
8. System.out.println("");
9. System.out.println("        handle(" + count + ", " + c + ") END");
10. }
11. **private** **void** slowly() {
12. **try** {
13. Thread.sleep(100);
14. } **catch** (InterruptedException e) {
15. }
16. }
17. }

**Java代码  [收藏代码](javascript:void())**

1. **public** **class** Main {
2. **public** **static** **void** main(String[] args) {
3. System.out.println("main BEGIN");
4. Host host = **new** Host();
5. host.request(10, 'A');
6. host.request(20, 'B');
7. host.request(30, 'C');
8. System.out.println("main END");
9. }
10. }

利用这个模式主线程不会等待执行任务的返回,就立即返回.   
对于网络分布式系统中不需要返回结果的IO流效率有很大的提高.还有一些比较耗时的方法也很有帮助.经典!   
  
适合环境:   
1.提升响应性,降低延迟时间   
2.适合在操作顺序无所谓时使用   
3.不需要返回值的时候   
4.应用在服务器的制作   
5.调用方法+启动线程--->传送消息

### 8、[woker thread pattern](http://pan-java.iteye.com/blog/423319)

工作区(核心)

**Java代码  [收藏代码](javascript:void())**

1. **public** **class** Channel {
2. //允许最多请求数
3. **private** **static** **final** **int** MAX\_REQUEST = 100;
4. //请求容器
5. **private** **final** Request[] requestQueue;
6. **private** **int** tail;  // 下一个putRequest的地方
7. **private** **int** head;  // 下一个takeRequest的地方
8. **private** **int** count; // Request的数量
10. //工作线程组
11. **private** **final** WorkerThread[] threadPool;
13. **public** Channel(**int** threads) {
14. //初始请求容器
15. **this**.requestQueue = **new** Request[MAX\_REQUEST];
16. **this**.head = 0;
17. **this**.tail = 0;
18. **this**.count = 0;
19. //初始化工作线程组
20. threadPool = **new** WorkerThread[threads];
21. **for** (**int** i = 0; i < threadPool.length; i++) {
22. threadPool[i] = **new** WorkerThread("Worker-" + i, **this**);
23. }
24. }
25. //工作线程开始工作
26. **public** **void** startWorkers() {
27. **for** (**int** i = 0; i < threadPool.length; i++) {
28. threadPool[i].start();
29. }
30. }
32. //添加请求
33. **public** **synchronized** **void** putRequest(Request request) {
34. //当容器缓冲请求数大于容器容量时等待
35. **while** (count >= requestQueue.length) {
36. **try** {
37. wait();
38. } **catch** (InterruptedException e) {
39. }
40. }
41. requestQueue[tail] = request;
42. //获取下一次放入请求的位置
43. tail = (tail + 1) % requestQueue.length;
44. count++;
45. notifyAll();
46. }
48. //处理请求
49. **public** **synchronized** Request takeRequest() {
50. //当请求数为空时等待
51. **while** (count <= 0) {
52. **try** {
53. wait();
54. } **catch** (InterruptedException e) {
55. }
56. }
57. Request request = requestQueue[head];
58. //获取下一次获取请求的位置
59. head = (head + 1) % requestQueue.length;
60. count--;
61. notifyAll();
62. **return** request;
63. }
64. }

请求对象

**Java代码  [收藏代码](javascript:void())**

1. **public** **class** Request {
2. **private** **final** String name; //  委托者
3. **private** **final** **int** number;  // 请求编号
4. **private** **static** **final** Random random = **new** Random();
5. **public** Request(String name, **int** number) {
6. **this**.name = name;
7. **this**.number = number;
8. }
9. **public** **void** execute() {
10. System.out.println(Thread.currentThread().getName() + " executes " + **this**);
11. **try** {
12. Thread.sleep(random.nextInt(1000));
13. } **catch** (InterruptedException e) {
14. }
15. }
16. **public** String toString() {
17. **return** "[ Request from " + name + " No." + number + " ]";
18. }
19. }

工作线程

**Java代码  [收藏代码](javascript:void())**

1. **public** **class** WorkerThread **extends** Thread {
2. **private** **final** Channel channel;
3. **public** WorkerThread(String name, Channel channel) {
4. **super**(name);
5. **this**.channel = channel;
6. }
7. **public** **void** run() {
8. **while** (**true**) {
9. Request request = channel.takeRequest();
10. request.execute();
11. }
12. }
13. }

放入请求线程

**Java代码  [收藏代码](javascript:void())**

1. **public** **class** ClientThread **extends** Thread {
2. **private** **final** Channel channel;
3. **private** **static** **final** Random random = **new** Random();
4. **public** ClientThread(String name, Channel channel) {
5. **super**(name);
6. **this**.channel = channel;
7. }
8. **public** **void** run() {
9. **try** {
10. **for** (**int** i = 0; **true**; i++) {
11. Request request = **new** Request(getName(), i);
12. channel.putRequest(request);
13. Thread.sleep(random.nextInt(1000));
14. }
15. } **catch** (InterruptedException e) {
16. }
17. }
18. }

测试类

**Java代码  [收藏代码](javascript:void())**

1. **public** **class** Main {
2. **public** **static** **void** main(String[] args) {
3. Channel channel = **new** Channel(5);   // 工作线程的數量
4. channel.startWorkers();
5. **new** ClientThread("Alice", channel).start();
6. **new** ClientThread("Bobby", channel).start();
7. **new** ClientThread("Chris", channel).start();
8. }
9. }

1.启动线程是繁重的操作   
  如果可以把自己的工作交给别人做,自己的可以去做其他事情.线程也一亲.如果可以把工作交给其他线程,自己就可以继续进到下一个工作.这是Thread-Per-Message Pattern 的主题.   
  
2.控制承载量   
Worker Thread 还有一个主题.就是承载量的控制.   
Worker 参与者的数量可以根据设置大小.   
  
3. invocation(启动方法)与execution(执行方法)分离   
    提高响应性   
    控制实行顺序   
    可取消和可重复执行   
    分散处理的第一步

### 9、[future](http://pan-java.iteye.com/blog/423365)

数据接口

**Java代码  [收藏代码](javascript:void())**

1. **public** **interface** Data {
2. **public** **abstract** String getContent();
3. }

真正需要的数据对象

**Java代码  [收藏代码](javascript:void())**

1. **public** **class** RealData **implements** Data {
2. **private** **final** String content;
3. **public** RealData(**int** count, **char** c) {
4. System.out.println("        making RealData(" + count + ", " + c + ") BEGIN");
5. **char**[] buffer = **new** **char**[count];
6. **for** (**int** i = 0; i < count; i++) {
7. buffer[i] = c;
8. **try** {
9. Thread.sleep(100);
10. } **catch** (InterruptedException e) {
11. }
12. }
13. System.out.println("        making RealData(" + count + ", " + c + ") END");
14. **this**.content = **new** String(buffer);
15. }
16. **public** String getContent() {
17. **return** content;
18. }
19. }

//异步数据

**Java代码  [收藏代码](javascript:void())**

1. **public** **class** FutureData **implements** Data {
2. **private** RealData realdata = **null**;
3. **private** **boolean** ready = **false**;
5. **public** **synchronized** **void** setRealData(RealData realdata) {
6. //如果已经准备好直接返回不用再设置
7. **if** (ready) {
8. **return**; // balk
9. }
10. **this**.realdata = realdata;
11. //设置值已准备好
12. **this**.ready = **true**;
13. //唤醒等待线程
14. notifyAll();
15. }
17. **public** **synchronized** String getContent() {
18. //如没有准备好,线程等待
19. **while** (!ready) {
20. **try** {
22. wait();
23. } **catch** (InterruptedException e) {
24. }
25. }
26. **return** realdata.getContent();
27. }
28. }

获取数据代理类

**Java代码  [收藏代码](javascript:void())**

1. **public** **class** Host {
2. **public** Data request(**final** **int** count, **final** **char** c) {
3. System.out.println("    request(" + count + ", " + c + ") BEGIN");
5. // (1) 建立FutureData的实体
6. **final** FutureData future = **new** FutureData();
8. // (2) 为了建立RealData的实体，启动新的线程
9. **new** Thread() {
10. **public** **void** run() {
11. RealData realdata = **new** RealData(count, c);
12. future.setRealData(realdata);
13. }
14. }.start();
16. System.out.println("    request(" + count + ", " + c + ") END");
18. // (3) 取回FutureData实体，作为传回值
19. **return** future;
20. }
21. }

//测试类

**Java代码  [收藏代码](javascript:void())**

1. **public** **class** Main {
2. **public** **static** **void** main(String[] args) {
3. System.out.println("main BEGIN");
4. Host host = **new** Host();
5. //请求数据
6. Data data1 = host.request(10, 'A');
7. Data data2 = host.request(20, 'B');
8. Data data3 = host.request(30, 'C');
10. System.out.println("main otherJob BEGIN");
11. **try** {
12. Thread.sleep(2000);
13. } **catch** (InterruptedException e) {
14. }
15. System.out.println("main otherJob END");
17. //不需要直接等待返回,可以同时去执行别的任务
18. System.out.println("data1 = " + data1.getContent());
19. System.out.println("data2 = " + data2.getContent());
20. System.out.println("data3 = " + data3.getContent());
21. System.out.println("main END");
22. }
23. }

1.异步方法调用返回值   
2.分离 "准备返回值"与"使用返回值"   
3.变形--不让人等待的Future参与者   
4.变形--会改变的Future参与者