**JAVA CORE BASIC**

1. Kỹ thuật lập trình căn bản
2. Download và cài đặt công cụ lập trình (Nếu là java thì dùng eclipse, hoặc myeclipse, hoặc net bean)
3. Các kiểu dữ liệu

|  |  |
| --- | --- |
| Kiểu dữ liệu | Kích thước |
| Boolelan | 1 |
| Char | 2 |
| Byte | 1 |
| Short | 2 |
| Int | 4 |
| Long | 8 |
| Float | 4 |
| Double | 8 |

1. Các câu lệnh if, for, while, do-while, case switch… Cho ví dụ minh họa, sự khác nhau.

-Mệnh đề if dùng để kiểm tra giá trị dạng boolean của điều kiện , điều kiện kiểm tra của mệnh đề if trả về giá trị true hoặc false.

Có kiểu mệnh đề if else trong java : if , if-else , if-else-if

-for dùng để lặp lại một chương trình , một phần chương trình trong một sô lần cho trước

Cấu trúc cơ bản của câu lệnh for :

for ( khởi tạo biến , kiểm tra điều kiện biến , tăng giảm biến ) ,

Đối với mảng hay collection trong java , có thể dùng for-each để lặp qua các phần tử :

For ( type var : array )

Ta có thể gán nhãn vòng lặp for , điều này là một cách đặt tên cho vòng lặp , giúp chủ động thực hiện các câu lệnh break , continue dễ dàng

Nhãn vòng lặp :: for ( khởi tạo biến , kiểm tra điều kiện biến , tăng giảm biến ) ,

* Khác nhau giữa while , do-while :

Vòng lặp while luôn kiểm tra điều kiện tại đầu vòng lặp trước khi thực thi đoạn mã trong khi do-while , chương trình trong do-while sẽ luôn được chạy 1 lần trước khi kiểm tra điều kiện

* Case switch : dùng để thực hiện 1 hoặc nhiều khối lệnh từ điều kiện
* Continue / break : hai từ khóa này được dùng trong vòng lặp :

+ , continue : khi gặp từ khóa này, vòng lặp hiện tại sẽ bị hủy bỏ , chương trình sẽ tiếp tục khởi tạo 1 vòng lặp mới để tiếp tục

+ break : khi găp từ khóa này , chương trình sẽ thoát khỏi vòng lặp , và tiếp tục với các câu lệnh khác sau vòng lặp.

1. Một số giải thuật làm việc với mảng: sắp xếp (các loại sắp xếp cơ bản), thêm, xóa 1 phần tử trong mảng, …
2. Streams Input/Out put, làm việc với file mức cơ bản.
3. Lập trình hướng đối tượng (phần này cần chi tiết và rõ ràng, có ví dụ cụ thể, làm thành tài liệu)
4. Các khái niệm: Đối tượng là gì? Class là gì?

Đối tượng là một thực thể có trạng thái , hành vi .

Class là một “ khuôn mãu “ để khởi tạo ra các đối tượng có thuộc tính chung , lớp sẽ quy định các thuộc tính và phương thức của các đối tượng được tạo từ lớp đó.

Tên public class được đặt trùng với tên file .java trong project . 1 file chỉ chứa duy nhất 1 public class , nếu class không phải public , 1 file có thể chứa nhiều class

* Access modifiers : từ khóa được dùng trước lớp , thuộc tính , phương thức nhằm chỉ định phạm vi truy cập của thuộc tính, phương thức , lớp .

Access modifiers : private ,public , default , protected.

+ , class

|  |  |
| --- | --- |
| Modifier | Decription |
| Public | Class có thể được truy cập từ tất cả các class khác trong project |
| Default | Đây là trường hợp trong class khi mình không chỉ đinh 1 modifier cho class . Class chỉ có thể truy cập trong cùng 1 package . |

+, attribute , method , constructor :

|  |  |
| --- | --- |
| Modifiers | Decription |
| Private | Chỉ có thể truy cập trong cùng 1 class |
| Public | Có thể truy cập mọi nơi |
| Default | Chỉ có thể truy cập trong cùng 1 package |
| Protected | Giống default , tuy nhiên có thể truy cập được tại những subclass. |

1. Các đặc điểm: (đa hình, thừa kế …), So sánh overloading và overriding, method overriding, abstract class, interface …

Oop có 4 tính chất cơ bản : đóng gói , kế thừa , đa hình , trừu tượng

-đóng gói : là việc che giấu những thông tinh thuộc tính của đối tượng , tùy chỉnh việc truy cập, sửa đối của các đối tượng khác tới nó .

Đạt được tính đóng gói thông qua access modifiers : private và cung cấp các hàm public getter / setter

Các hàm getter/setter sẽ cung cấp tùy chỉnh về truy cấp của đối tượng : write only , read only đồng thời cũng là nơi kiểm tra điều kiện trước khi việc thay đổi , đọc ghi thuộc tính diễn ra.

+, Kế thừa : một lớp có thể có toàn bộ những thuộc tính ,phương thức , constructor của 1 lớp khác => tái sử dụng code

Đạt được sự kế thừa trong java bằng từ khóa “extends” :

class ten\_lóp\_con extends ten\_lop\_cha

Trong java chỉ hỗ trợ đơn kế thừa : một lớp chỉ kế thừa từ 1 lớp khác

+ đa hình : một phương thức có thể được thực thi theo nhiều cách khác nhau.

Đạt được tính đa hình : overloading / override

|  |  |
| --- | --- |
| Overloading | Override |
| +, method signature của hai phương thức khác nhau  +, là việc định nghĩa thêm hành vi cho phương thức.  +, được thực hiện trong cùng 1 class  +, được quyết định tại thời điểm compiler time | +method signuture giống hệt nhau  + là việc định nghĩa lại hành vi method  + được dùng trong hai class có quan hệ kế thừa  + được quyết định thực hiện tại run time |

+ , Trừu tượng : là việc che dấu những thông tin quan trọng , chỉ hiển thị những thông tin quan trọng, che dấu những thông tin quan trọng với người dùng .

Tính trừu tượng giúp ltv tập trung vào những chức năng cốt lỗi, loại bỏ tính phức tạp của đối tượng , tăng tính mở rộng của hệ thống.

Đạt được tính trừu tượng dùng abstract , interface

Cách khai báo lớp, thuộc tính, phương thức, constructor, cách phân chia các class ?

Lớp được khai báo bằng từ khóa class , chúng ta có thể chỉ định các access modifier cho class : public , default .

Tên public class trùng với tên file .java. Một file .java chứa duy nhất 1 public class . 1 file .java có thể có nhiều defautl class.

1. Exception handling

Ngoại lệ là một sự kiện bất thường xảy ra trong thời gian runtime , làm gián đoạn luồng hoạt động bình thường của chương trình .

1. Khai báo ngoại lệ

Có 3 loại exception :

+,Checked Exception : lỗi xảy ra lúc Compile time

+,Unchecked Exception : Các ngoại lệ không đc kiểm tra tại thời điểm biên dịch, đc kiểm tra tại thời gian chạy

+Error : lỗi không thể phục hồi

Throwable là lớp gốc của hệ thống phân cấp ngoại lệ. Nó được kế thừa bởi 2 lớp con là Exception và Error

1. Cơ chế try-catch

* Là cơ chế để xử lí ngoại lệ

Chi tiết :

|  |  |
| --- | --- |
| Key word | Decription |
| Try | Chứa đoạn mã chứa ngoại lệ |
| Catch | Chứa đoạn mã dùng để xử lí ngoại lệ |
| Finally | Chứa đoạn mã luôn được thực thi |
| Throw | Ném ngoại lệ |
| Throws | Khai báo ngoại lệ |

1. Throw ngoại lệ, Chuyển tiếp ngoại lệ

Là ngoại lệ được ném ra , chủ yếu là là ngoại lệ của người dùng tự định nghĩa , chương trình sẽ dừng thực thi và Exception đó sẽ được catch , nếu ngoại lệ được throw ra nhưng không được catch thì phải khai báo nó trên lớp bằng từ khóa “throws”

Những nơi dùng phương thức được throws exception đó phải được try-catch nếu ko quá trình trên tiếp tục,

1. Lập trình xử lý với database
2. Cài đặt oracle (10G or 11G)
3. Sử dụng oracle: các câu lệnh cơ bản, table, store procedure, function, trigger, sequence, partition, job, schedule…
4. Kết nối với oracle, tự làm bài tập để test
5. Học cách làm quen và sử dụng file config trong các module: (sửa dụng 14, hoặc ở mức cơ bản có thể tự viết đọc file text)
6. Đóng gói chương trình, run file trên windows, linux …

git : <https://github.com/doanphung0402/javacore_xtel.git>

1. **Bài tập (*Về phần bài tập các tham số được lưu trữ trong các file cấu hình – cho phép truyền vào mà không cần build lại code, Viết trên console*) -done**
2. Viết chương trình nhập 1 số cho tới khi nhập đúng số cần nhập thì hiện thông báo thành công. Số lần nhập tối đa 5 lần nếu nhập sai thì sẽ dừng chương trình và hiện thông báo nhập lỗi.
3. Viết chương trình tính tiền điện, cho phép nhập số điện đã dùng trong tháng, giả thiết 100 số đầu tính giá 1000đ/1 số, kể từ 100 số tính giá 1500đ/số, kể từ 150 số tính giá 2000đ/số.
4. Sắp xếp mảng số nguyên 1 chiều gồm n phần tử cho trước được đọc ra từ file input.txt bằng phương pháp quick sort, giả thiết các số nguyên cách nhau bằng dấu cách hoặc xuống dòng
5. Viết chương trình console, thực hiện nhập thông tin n sinh viên (tên, giới tính, quê quán, tuổi) Sau khi nhập xong bấm enter để insert thông tin vào DB, giả thiết tên không được trùng nhau, id tự tăng.

**JAVA CORE ADVANCE**

1. Streams Input/out put, làm việc với file mức nâng cao (tự nghiên cứu sử dụng các thư viện nâng cao trong Java), ưu nhược điểm.

Làm việc với file trong java

Java I/O hay Input/Output trong javađược sử dụng để xử lý đầu vào và đầu ra trong java.

Java sử dụng khái niệm stream để làm cho hoạt động I/O nhanh hơn. Gói java.io chứa tất cả các lớp cần thiết cho hoạt động input và output.

Một stream là một dãy dữ liệu. Trong java, một stream bao gồm các byte. Nó được gọi là stream (dòng chảy) vì nó giống như một dòng nước chảy liên tục.

OutputStream và InputStream :

OutputStream

Ứng dụng Java sử dụng một output stream để ghi dữ liệu đến đích, nó có thể là một tệp tin, một mảng, thiết bị ngoại vi hoặc socket.

Lớp OutputStream là một lớp trừu tượng. Nó là super class của tất cả các lớp đại diện cho một output stream của các byte. Một output stream chấp nhận ouput các byte và gửi chúng đến một nơi có thể chứa.

Các phương thức của lớp OutputStream

|  |  |
| --- | --- |
| Method | Description |
| 1) public void write(int)throws IOException | được sử dụng để ghi một byte đến output stream hiện tại. |
| 2) public void write(byte[])throws IOException | được sử dụng để ghi một mảng các byte đến output stream hiện tại. |
| 3) public void flush()throws IOException | flush output stream hiện tại. |
| 4) public void close()throws IOException | được sử dụng để đóng output stream hiện tại. |

InputStream

Ứng dụng Java sử dụng một input stream để đọc dữ liệu từ một nguồn, nó có thể là một tệp tin, một mảng, thiết bị ngoại vi hoặc socket.

Lớp InputStream là một lớp trừu tượng. Nó là super class của tất cả các lớp đại diện cho một input stream của các byte.

Các phương thức của lớp InputStream

|  |  |
| --- | --- |
| Method | Description |
| 1) public abstract int read()throws IOException | Đọc byte kế tiếp của dữ liệu từ input stream. Nó trả về -1 khi đọc đến vị trí cuối tập tin. |
| 2) public int available()throws IOException | Trả về một ước tính về số byte có thể đọc được từ input stream hiện tại. |
| 3) public void close()throws IOException | được sử dụng để đóng input stream hiện tại. |

1, FileInputStream and FileOutputStream

Extends từ InputStream,OutputStream

Đọc , ghi file dưới dạng byte .

2, BufferedOutputStream và BufferedInputStream

Hoạt động như FileInputStream và FileOutputStream, tuy nhiên BufferedOutput(Input)Stream

Sử dụng thêm cơ chế bộ đệm tăng tốc độ đọc ghi file.

Mở rộng của lớp FileInputStream và FileOutputStream

3. ByteArrayOutputStream và ByteArrayInputStream

+, sử dụng để ghi dữ liệu chung vào nhiều file. Trong luồng này, dữ liệu được ghi vào mảng byte có thể được ghi vào nhiều stream sau đó.

+, giữ một bản sao của dữ liệu và chuyển tiếp nó đến nhiều stream.

+, Bộ đệm của ByteArrayOutputStream tự động tăng theo kích thước dữ liệ

1. Thread (chi tiết, thực hành ngay nếu học xong lý thuyết)

* Khái niệm
  + - Một thread là một quy trình con nhẹ , là đơn vị nhỏ nhất trong java , mỗi thread thực thi 1 công việc riêng biệt
    - Tại một thời điểm chỉ có duy nhất một luồng thực thi.
    - Java chính là một ngôn ngữ lập trình đa phân luồng (**multithreaded**) => có thể chia nhỏ nhiều hoạt động riêng biệt bên trong 1 ứng dụng lớn.
    - Các thread có thể chạy song song và biệt lập lẫn nhau.

Khi một luồng được thực thi, nó sẽ không thể được thực thi lại => lỗi IllgalThreadStateException.

Mỗi thread chạy trong 1 callstack ;

* Khởi tạo, có mấy cách khởi tạo, ưu nhược điểm

Có hai cách tạo 1 thread :

* + extends Thread
  + Implement Runable

|  |  |
| --- | --- |
| extends Thread | Implements runable |
| Kế thừa lớp Thread , cung cấp nhiều hàm tạo và phương thức để thao tác trên 1 thread.  Chỉ có thể kế thừa duy nhất 1 class Thread .  Luồng khởi chạy bởi class extends Thread cần khởi tạo đối tượng và gọi hàm start() để khởi chạy => mỗi luồng gắn duy nhất với 1 đối tượng , không thể cập nhật chung tài nguyên trong mỗi luồng | Triển khai interface runable , có duy nhất phương thức run() để khởi chạy luồng .  Tận dụng được tính chất đa kế thừa của interface trong java,=> cho phép chia sẻ 1 đối tượng cho nhiều Thread.  Dùng hàm tạo của Thread class khởi tạo luồng :   * Thread(String name) * Thread (Runable runable ,String name) * Thread(Runable runable) * Thread()   Hàm tạo nhận vào obj Runable => dùng chung 1 obj cho 1 nhóm luồng => dùng khi cằn cập nhật chung 1 tài nguyên cho 1 nhóm luồng . |

* Dùng runable khi :

+ tạo tính loosely-coupled trong ứng dụng

+ cần cập nhật chung 1 tài nguyên cho đối tượng

+ logic bớt phức tạp

* Thread Priority

Thread Scheduler trong java nhận nhiệm vụ điều phối hoạt động giữa các luồng,

Có hai yếu tố để Thread Scheduler quyết định khi nào 1 luồng đc thực thi : time of arrival , Priority

* + - * Priority : Mức độ ưu tiên của luồng , có giá trị từ 1-> 10 , priority càng cao mức độ ưu tiên được chạy càng lớn
      * Time of arrival : nếu hai luồng cùng mức độ ưu tiên => time of arrival của luồng sẽ được ưu tiên lựu chọn.

Priority có 3 constant : MIN\_PRIORITY =1 , MAX\_PRIORITY =10 , NORM\_PRIORITY =5 (default) .

Mặc định priority của thread là 5 , của main thread là 10

Dùng các phương thức getPriority/setPriority để lấy hoặc set độ ưu tiên cho thread.

* Vòng đời của 1 thread, các trạng thái

Các luồng có 5 trạng thái : new state , active state , waiting/blocked state ,timed waiting state , terminated state.

* New state : trạng thái đầu tiên khi khởi tọa của thread , tồn tại cho tới kho chương trình bắt đầu chạy , tại thời điểm này thread có thể chạy tại bất cứ khi nào đc gọi
* Active state : khi thread đang thực hiện tác vụ của mình .

+, runable state : thread được hệ thống cung cấp tài nguyên và thời gian , sẵn sàng để chạy

+, run state : thread chuyển từ runable xang running , thread thực thi công việc cua mình trong thời gian mà tiến trình cung cấp .

* Waiting / Blocked state : trạng thái chờ của 1 thread , thread vào trạng thái waiting hay block , hay là nó sẽ đợi vô hạn để chờ 1 tác vụ hay tài nguyên nào đó. Và khi hệ thống ycau nó sẽ chuyển về trạng thái runable .

Khác nhau : waiting của thread do bản thân thread tụ áp đặt để đạt được sự đồng bộ với các luồng khác.

Block : do cơ chế của các tiến trình áp đặt giàm giải quyết việc chia sẻ tài nguyên găng trong hệ thống.

* Time waiting : là trạng thái waiting của luồng , thread sẽ về trạng thái waiting trong một khoảng thời gian xác định và sẽ chờ thông báo của những thread khác để chuyển về trạng thái runable .
* Terminated : là trạng thái cuối vòng đời của thread , khi nó hoàn thành xong tất cả các tác vụ của mình .
* Dealock, khái niệm, ví dụ :

Phương thức đồng bộ trong java

+ Điều phối tiến trình với tài nguyên găng. => chỉ chdo phép 1 luồng truy cập tài nguyên tại một thời điểm => tính nhất quán dữ liệu, tránh xung đột

Đồng bộ hóa là việc xắp sếp thứ tự các thread khi truy xuất vào cùng 1 đối tượng sao cho không có sự xung đột dữ liệu.

Giải pháp hỗ trợ :

Java sử dụng 1 “Monitor “ dùng giám sát , hỗ trợ cho việc đồng bọ hóa các luồng.

Tại 1 thời điểm chỉ có duy nhất 1 thread được truy cập vào “monitor “ , các thread còn lại phải chờ đến khi thread này ra khỏi monitor .

* 2 phương pháp Thread synchronized : Mutual Exclusive , Coopration

Mutual Exclusive : Điều phối tiến trình qua tài nguyên găng , đây là giải pháp đưa thread vào monitor : synchronized method , synchronized block

* + - * Synchronized method: sử dụng non- access modifiers : synchronized => khi một thread gọi obj chứa synchronized method thì obj sẽ block và sẽ giải phóng khi thread kết thúc.

Ex : synchronized void methodNam() ;

* + - * synchronized block : dùng để thực hiện trên một đoạn mã logic bất kì , ko phải cả method .

Ex : synchronized ( object reference expression || ) { // code }

Phạm vi synchronized block nhỏ hơn synchronized method

Synchronized block được dùng để đồng bộ trên bất ki tài nguyên cụ thể nào của phương thức , khóa chính xác trên đoạn code ta cần .

Static synchronized :

Coopratione : Cơ chế giao tiếp liên quá trình bằng cách sử dụng phương thức wait(), notify(), notifyAll() trên đối tượng chia sẻ

Các method wait(), notify(),notifyAll() chỉ được gọi bên trong 1 synchronized method , synchronized block

+, wait() || wait(long timeout) : đưa thread vào trạng thái sleeping(), chờ đến khi có notify() của một thread khác , hoặc chờ trong 1 timeout cụ thể

+, notify() : đánh thức thread ở trạng thái sleeping đang chờ trên monitor khi được gọi wait (), nếu có nhiều thread thì 1 trong số chúng sẽ được đánh thức.

+, notifyAll() : đánh thức tất cả các threafd bị slepping bởi wait() , thread có priority cao nhất sẽ chạy đầu tiên.

Ex : /InterThread.java

Dead lock (khóa chết hay bế tắc) : là tình huống xảy ra khi hai hay nhiều tiến trình chờ đợi lẫn nhau , và rơi vào trạng thái chờ đợi vòng tròn .

Ví dụ :

Ta có hai tài nguyên là 2 biến như sau: Ảnh có chứa văn bản

Mô tả được tạo tự động

Ta có 2 Thread cùng truy cập vào hai tài nguyên như sau :

Thread 1 :

Ảnh có chứa văn bản

Mô tả được tạo tự động

Luồng 2 :

Ảnh có chứa văn bản

Mô tả được tạo tự động

Hai luồng dùng khởi tạo từ obj Thread , thực hiện synchronized block :

Thread 1 khởi chạy phương thức run() , thực hiện “block “ trên resource1 sau đó chờ 100ms để tiếp tục “block” resource2 , lúc này thread2 khởi chạy ( giải sử chạy sau thread1) , thhread 2 thực hiện synchronized block trên resource2 , trong đó cũng chờ 100ms để block resource1 , tuy nhiên lúc này resource1 đang bị thread1 block và thread1 đang chờ resource2 đang bị “block” trên thread2

=>Cả hai thread đang cùng chờ đợi “tài nguyên “ lẫn nhau => trương trình bị treo

Ảnh có chứa văn bản

Mô tả được tạo tự động

Giair pháp :

* + - Tránh “block” lồng nhau
    - Tránh “block” khi không cần thiết
    - Sử dụng method Join() của thread , bắt 1 thread chờ kết thúc của thread khác => đảm bảo tính tuần tự

* Timer, Time Task, Schedule … Khái niệm, cho ví dụ

Timer và Time Task là “ java util class” được sử dụng để lên lịch cho một công việc và chạy nó trong background với 1 Thread.

Cụ thể : TimerTask là những task cần thực hiện ; Timer là trình thực thi

Cả TimerTask và Thread đều thực thi không đồng bộ , nhưng TimeTask được lên lịch để thực thi ở các khoảng thời gian cố định trong khi thread thì không .

Thread được thực thi bất cứ khi nào nó trên hàng đơi và có lượt chạy. Trong khi TimerTask chỉ có đủ điều kiện chạy trong các khoảng thời gian cố định mà đã được lên lịch để chạy

Cách hoạt động của Thread và Timer giống hệt nhau

Schedule - Trình lên lịch của Thread : Bởi vì trong 1 thời điểm chỉ có 1 thread được chạy vì vậy Thread Scheduler sẽ lên lịch để thực th3i các luồng.

Có hai yếu có quyết đinh cho Thread Schedule quyết định : priority + time of arival

1. Lập trình mạng
2. Socket

Cưng cấp cơ chế truyền thông giữa hai máy tính sử dụng TCP .Một máy khách tạo kết nối để kết nối tới máy chủ

Java cung cấp lớp java.net.Socket để thao tác với socket

1. Tcp , ud: khái niệm, cơ chế, ví dụ

TCP – Transmission Control Protocol , được sử dụng qua giao thức internet => là phương thức truyền tin hướng kết nối, tin cậy có kiểm soát , cho phép giao tiếp tin cậy giữa hai ứng dụng

Hoạt động :

Tcp thường được dùng với ip trở thành bộ giao thức tcp/ip.

Dưa trên mô hình client -server.

Truyền dữ liệu hai chiều

* + Hướng kết nối : bắt tay 3 bước => tính bảo mật
  + Hoạt động dựa trên cơ chế báo nhận => đảm bảo toàn vẹn dữ liệu =>cung cấp cơ chế phục hồi dữ liệu bị mất
  + Kiểm soat luồng , kiểm soát truyền tải tránh tác nghẽn
  + Đánh số hiệu gói tin truyền đi => đảm bảo thứ tự gói tin nhận được
  + Hỗ trợ phuwogn thức checksum kiểm tra tính toàn vẹn cuả dữ liệu nhận được
  + Hỗ trợ truyền song công .
* Tcp dùng trong trường hợp truyền tải dữ liệu có độ tin cậy cao nhưng không yêu cầu tốc độ truyền tải .

Ví dụ :

1. UDP: khái niệm, cơ chế, ví dụ

UDP–user datagram protocol , một phương thức truyền dữ liệu giữa các ứng dụng , la phương thức truyền tin không tin cậy , không kiểm soát , hướng kết nối.

Hoạt động : UDP không có thiết lập kết nối trước khi truyền dữ liệu , khi có dữ liệu ,dữ liệu sẽ được đóng gói vào các gói datagram và lạp tức được gửi đi.

Cấu trúc UDP datagram : 64bit :

UDP checksum (16bit ) : kiểm soát lỗi hủy bỏ UDP datagram lỗi mà không cần thông báo với bên gửi.

Source Port : 16bit : nơi gửi datagram .

Destination Port 16bit : địa chỉ nơi nhận

Length (16bit) : độ đài header vằ data

Checksum : dùng kiểm tra lỗi của gói tin

Đặc điểm :

Tiêu đề gói dữ liệu nhỏ

Không cần thiết lập kết nối

Không duy trì trạng thái kết nối

Không có cơ chế báo nhận

Không xắp sếp tuần tự các gói tin

Không có cơ chế kiểm soát

Ưu điểm :

Không mất thowqif gian thiết lập kết nỗi

Nhanh, ít thời gian xử lí

Nhược điểm :

Không có cơ chê kiểm soát mất mát => độ tin cậy thấp

* Sử dụng trong các dịch vụ thời gian thực , không cần độ chính xác dữ liệu

1. So sánh tcp và udp

Giống nhau :

Hai giao thức dùng để truyền dữ liệu – gói tin qua internet

Xây dựng trên nền giao thức ip

Hai giao thức hoạt đông trên tầng giao vận

Khác nhau :

|  |  |
| --- | --- |
| TCP | UDP |
| * Hướng kết nối * Độ tin cậy cao * Gửi dữ liệu dạng luồng byte * Không cho phép mất gói tin * Đảm bảo việc nhận đúng thứ tự gói * Tốc độ truyền thấp | * Không hướng kết nối * Độ tin cậy thấp * Gửi dũ liệu dưới dạng datagram * Cho phép mất gói * Không đảm bảo nhận đúng thú tự gói tin * Tốc dộ truyền nhanh * Không kiểm soát tốc độ truyền |

1. MultiThread
2. Khái niệm

Là quá trình đồng thời thực thi nhiều luồng => sử dụng đạt được đa nhiệm

Các luồng sử dụng bộ nhớ dùng chung , độc lập.

1. Tìm hiểu Queue là gì? các loại? cách làm việc với queue

Queue là một cấu trúc dữ liệu , các phần tử lưu trữ trong queue được lưu trữ và truy cập theo phương pháp : first in first out (FIFO)

Queue trong java là 1 interface , Các class trong java triển khai Queue trong java : PriorityQueue , LinkedList , ArrayDeque.

Queue extend từ interface Collection , có một số phuong thức thường sử dụng với queue là

Add() : chèn phần tử mới vào hàng đợi , return “ true” nếu thành công, throw Exception nếu thất bại

Offer() : giông add() , nhưng nếu thất bại return false.

Element() : trả về phần tử đầu tiên , nếu queue trống ném ngoại lệ

Peek() : trả về phần tử đầu tiên , nếu không có return null ;

Remove() : xóa phần tử đầu tiên khỏi hàng đợi , nếu ko có throw exception

Poll() : giống remove , nếu không có trả về null ;

Boolean isEmpty() : kiểm tra hàng đợi có trống hay không

Khởi tạo queue :

+, Với LinkedList: Queue<Obj> queue = new LinkedList<Obj>();

+,Với PriorityQueue : đây là hàng đợi , nhưng việc xử lí các phần tủ dựa trên mức độ ưu tiên.

Các phần tử trong queue có mã ASCII lớn sẽ có độ ưu tiên càng cao ., các phần tử có độ ưu tiên lớn sẽ nằm ở đầu hàng đợi

Queue<obj> queue = new PriorityQueue<>();

Queue có ba loại :

Deque : là một “ double-ended queue “ \_ hàng đợi hai chiều , có thể hoạt động như 1 queue hoặc như 1 stack

Deque là 1 interface : interface deque extends Queue

Khởi tạo : Dequeue dequeue = new ArrayDequeue<Obj>()

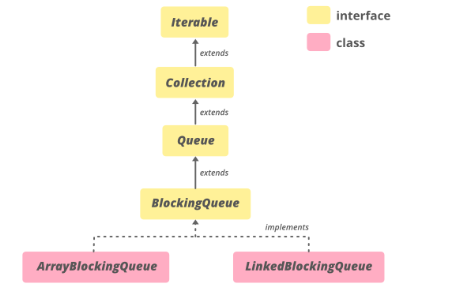
Các phương thức được sử dụng dequeu

|  |  |
| --- | --- |
| [add(element)](https://www.geeksforgeeks.org/deque-add-method-in-java/) | This method is used to add an element at the tail of the queue. If the Deque is capacity restricted and no space is left for insertion, it returns an IllegalStateException. The function returns true on successful insertion. |
| [addFirst(element)](https://www.geeksforgeeks.org/deque-addfirst-method-in-java-with-examples/) | This method is used to add an element at the head of the queue. If the Deque is capacity restricted and no space is left for insertion, it returns an IllegalStateException. The function returns true on successful insertion. |
| [addLast(element)](https://www.geeksforgeeks.org/deque-addlast-method-in-java/) | This method is used to add an element at the tail of the queue. If the Deque is capacity restricted and no space is left for insertion, it returns an IllegalStateException. The function returns true on successful insertion. |
| [contains()](https://www.geeksforgeeks.org/deque-contains-method-in-java/) | This method is used to check whether the queue contains the given object or not. |
| [descendingIterator()](https://www.geeksforgeeks.org/deque-descendingiterator-method-in-java/) | This method returns an iterator for the deque. The elements will be returned in order from last(tail) to first(head). |
| [element()](https://www.geeksforgeeks.org/deque-element-method-in-java/) | This method is used to retrieve, but not remove, the head of the queue represented by this deque. |
| [getFirst()](https://www.geeksforgeeks.org/deque-getfirst-method-in-java/) | This method is used to retrieve, but not remove, the first element of this deque. |
| [getLast()](https://www.geeksforgeeks.org/deque-getlast-method-in-java/) | This method is used to retrieve, but not remove, the last element of this deque. |
| [iterator()](https://www.geeksforgeeks.org/deque-iterator-method-in-java/) | This method returns an iterator for the deque. The elements will be returned in order from first (head) to last (tail). |
| [offer(element)](https://www.geeksforgeeks.org/deque-offer-method-in-java/) | This method is used to add an element at the tail of the queue. This method is preferable to add() method since this method does not throws an exception when the capacity of the container is full since it returns false. |
| [offerFirst(element)](https://www.geeksforgeeks.org/deque-offerfirst-method-in-java/) | This method is used to add an element at the head of the queue. This method is preferable to addFirst() method since this method does not throws an exception when the capacity of the container is full since it returns false. |
| [offerLast(element)](https://www.geeksforgeeks.org/deque-offerlast-method-in-java/) | This method is used to add an element at the tail of the queue. This method is preferable to add() method since this method does not throws an exception when the capacity of the container is full since it returns false. |
| peek() | This method is used to retrieve the element at the head of the deque but doesn’t remove the element from the deque. This method returns null if the deque is empty. |
| peekFirst() | This method is used to retrieve the element at the head of the deque but doesn’t remove the element from the deque. This method returns null if the deque is empty. |
| peekLast() | This method is used to retrieve the element at the tail of the deque but doesn’t remove the element from the deque. This method returns null if the deque is empty. |
| poll() | This method is used to retrieve and remove the element at the head of the deque. This method returns null if the deque is empty. |
| pollFirst() | This method is used to retrieve and remove the element at the head of the deque. This method returns null if the deque is empty. |
| pollLast() | This method is used to retrieve and remove the element at the tail of the deque. This method returns null if the deque is empty. |
| pop() | This method is used to remove an element from the head and return it. |
| push(element) | This method is used to add an element at the head of the queue. |
| removeFirst() | This method is used to remove an element from the head of the queue. |
| removeLast() | This method is used to remove an element from the tail of the queue. |
| size() | This method is used to find and return the size of the deque. |

BlockingQueue : giống queue nhưng hỗ trợ thêm tính năng “ flow control” \_kiểm soát luồng. Cung cấp thêm tính năng để “block “ khi queue đầy hoặc trống, an toàn khi sử dụng với các thao tác đa luồng đảm bảo tính nhất quán dữ liệu

Blockingqueue không hỗ trợ giá trị “null” => throw NullPulumException

Sử dụng : LinkedListBlockingQueue , PriorityBlockingQueue để triển khai



Tạo BlockingQueue :

BlockingQueue<?> objectName = new LinkedBlockingQueue<?>(capacity);

OR

LinkedBlockingQueue<?> objectName = new LinkedBlockingQueue<?>(capacity);

//capacity : dung lượng tối đa của queue .

Một số thao tác cơ bản trên LinkedBlockingDeque, như tạo đối tượng, thêm phần tử, xóa phần tử và sử dụng trình vòng lặp để duyệt qua LinkedBlockingQueue.

BlockingDeque : <tương tự >

1. Tìm hiểu và làm ví dụ với Thread Pool: Khái niệm, cách sử dụng, ưu nhược điểm …

Thread pool là 1 nhóm các luồng đang đợi hoặc một nhóm các luồng được sử dụng nhiều lần

Khi cần sử dụng , 1 thread trong thread pool được lấy ra và được running , khi hoàn thành thread lại được được đưa vào pool

Thread pool tạo râ nhiều thread để cùng xử lí 1 công việc duy nhất

* ưu điểm :

Giới hạn số lượng thread được sinh ra trong ưng dụng tại 1 thời điểm => cải thiện hiệu suất , bộ nhớ so với việc khởi tạo từng thread bằng tù khóa new

Kiểm soát hiệu năng , ngăn chặn quá tải : có thể giới hạn các thread chạy đồng thời trong 1 thread pool , khi thread trong pool đều đang làm việc và có task mới đến => task sẽ được thêm vào task queue – blockind queue để chờ 1 thread có sẵn.

Hoạt động :

Ảnh có chứa bàn

Mô tả được tạo tự động

Một số thread pool có sẵn :

Cached thread pool : giữ 1 số thread alive , sẵn sàng tạo mới thread nếu cần.

Fixed thread pool: giới hạn số lượng tối đa của các Thread được tạo ra để thực thi các task (nhiệm vụ). Các task khác đang chờ trong hàng đợi (BlockingQueue).

Single-threaded pool: chỉ giữ một Thread thực thi một nhiệm vụ một lúc.

java.util.concurrent : cung cấp các API hỗ trợ thread pool .

Executor : là đối tượng chịu tránh nghiệm quản lí thread trong thread pool

Java Concurrency API định nghĩa 3 interfaces cơ bản sau cho các Executor:

Executor: là interface cha của tất cả Executor. Nó xác định chỉ một phương thực excute(Runnable).

ExecutorService: là một Executor cho phép theo dõi tiến trình của các tác vụ trả về giá trị (Callable) thông qua đối tượng Future, và quản lý việc kết thúc các luồng. Các phương thức chính của nó bao gồm submit() và shutdown().

ScheduledExecutorService: là một ExecutorService có thể lên lịch cho các tác vụ để thực thi sau một khoảng thời gian nhất định, hoặc để thực hiện định kỳ. Các phương thức chính của nó là schedule(), scheduleAtFixedRate() and scheduleWithFixedDelay().

Chúng có thể tạo một Executor bằng cách sử dụng một trong các phương thức được cung cấp bởi lớp tiện ích Executors như sau:

* newSingleThreadExecutor(): trong ThreadPool chỉ có 1 Thread và các task (nhiệm vụ) sẽ được xử lý một cách tuần tự.
* newCachedThreadPool(): trong ThreadPool sẽ có nhiều Thread và các nhiệm vụ sẽ được xử lý một cách song song. Các Thread cũ sau khi xử lý xong sẽ được sử dụng lại cho nhiệm vụ mới. Mặc định nếu một Thread không được sử dụng trong vòng 60 giây thì Thread đó sẽ bị tắt.
* newFixedThreadPool(int n): trong ThreadPool sẽ được cố định các Thread. Nếu một nhiệm vụ mới được đưa vào mà các Thread đều đang “bận rộn” thì nhiệm vụ đó sẽ được gửi vào Blocking Queue và sau đó nếu có một Thread đã thực thi xong nhiệm vụ của nó thì nhiệm vụ đang ở trong Queue đó sẽ được push ra khỏi Queue và được Thread đó xử lý tiếp.
* newScheduledThreadPool(int corePoolSize): tương tự như newCachedThreadPool() nhưng sẽ có thời gian delay giữa các Thread.
* newSingleThreadScheduledExecutor(): tương tự như newSingleThreadExecutor() nhưng sẽ có khoảng thời gian delay giữa các Thread.

Nhược điểm :

Deadlock : tất cả các luồng đang thực thi đang chờ kết quả từ các luồng bị chặn và đang chờ trong hàng đợi do không có sẵn luồng để thực thi.

Thread Leakage :  Rò rỉ luồng xảy ra khi một luồng đang bị xóa khỏi nhóm để thực thi một tác vụ nhưng không quay lại luồng đó sau khi hoàn thành tác vụ

Resource Thrashing: Rất nhiều thời gian bị lãng phí trong việc chuyển ngữ cảnh giữa các luồng khi kích thước của nhóm luồng rất lớn. Bất cứ khi nào có nhiều luồng hơn số lượng tối ưu có thể gây ra vấn đề chết đói và dẫn đến tình trạng cạn kiệt tài nguyên.

Thế nào là đồng bộ và bất đồng bộ, làm thế nào để đồng bộ khi nhiều thread cùng sử dụng 1 tài nguyên? Các vấn đề cần xử lý khi chương trình có nhiều thread?... (Cần đọc nhiều về multi threading) // <ở trên >

*Hướng dẫn: Học về synchronize, lock, unlock, cơ chế wait, notify …, Phần này tương đối phức tạp cần nghiên cứu và làm nhiều bài tập thì càng tốt.*

1. Logger1 ngày

* Mục đích, vai trò cho việc logging?

Lớp Logger cung cấp phương thức để viết nhât kí

Vai trò :

Nắm bắt luồng chạy của chương trinh

Giúp phát hiện , debug lỗi trong quá trình chạy.

* Tự tạo ra log, viết chương trình và ghi log, thư viện hỗ trợ log?

Khởi tạo logger :

Logger logger = Logger.getLogger(logger\_name); => dùng để tạo logger hay trả về logger cùng tên.

Or

Logger logger = Logger.getLogger(Myclass.class.getName());

-Có 7 loại logger tương ứng 7 cấp độ quan trọng

|  |  |
| --- | --- |
| Cấp độ nhật ký (theo thứ tự giảm dần) | Sử dụng |
| SEVERE | thất bại nghiêm trọng |
| WARNING | thông báo cảnh báo, một vấn đề tiềm ẩn |
| INFO | thông tin thời gian chạy chung |
| CONFIG | thông tin cấu hình |
| FINE | thông tin chung về nhà phát triển (theo dõi thông báo) |
| FINER | thông tin chi tiết về nhà phát triển (thông báo theo dõi) |
| FINEST | thông tin nhà phát triển rất chi tiết (theo dõi thông báo) |
| OFF | tắt ghi nhật ký cho tất cả các cấp (không ghi gì) |
| ALL | bật ghi nhật ký cho tất cả các cấp (nắm bắt mọi thứ) |

-đặt cấp độ cho Log : logger.setLog(Level.Logger\_level);

Để ghi log ta dùng phương thức log ex : log.warning(“log\_message”);

1. **Bài tập thực hành**
2. Viết 1 chương trình chạy real-time, ghi các số nguyên random ra file output.txt . Dừng chương trình khi gõ lệnh stop trên cửa sổ chương trình.
3. Viết 1 chương trình chạy real-time, cứ n giây in 1 số nguyên random ra mà hình . Dừng chương trình sau n phút.
4. Viết chương trình giải quyết bài toán producer & consumer với các yêu cầu sau:

* Có một message queue chứa các message, cấu trúc message là tùy chọn. Message queue có size giới hạn.
* Một thread đóng vai trò producer: producer định kỳ sẽ tạo ra một message và đưa vào message queue, nếu msgq đã full thì thread sẽ phải đợi cho tới khi msgq không ở trong trạng thái full và tiếp tục tạo message mới đưa vào queue.
* Một thread đóng vai trò consumer: mỗi lần sẽ lấy ra 1 message từ message queue và in message ra màn hình, nếu message queue đang empty thì sẽ phải đợi cho tới khi có message trong queue để xử lý.

1. Viết chương trình đơn giản gửi liên tục , mỗi lần 1 chuỗi tự sinh ngẫu nhiên (Random theo cấu trúc tự tạo) từ máy A, và gửi tới 1 máy tính B khác bằng socket, Máy B nhận được sẽ in ra màn hình kết quả nhận được. Yêu cầu xử lý ngoại lệ trong quá trình gửi nhận (có thể test bằng việc rút dây mạng trong quá trình gửi nhận), ghi ra log file nếu xảy ra lỗi, đưa vào file cấu hình các tham số: ip, port, connection time-out, send time-out, receive time-out … (bài toán có thể mở rộng ra: viết chương trình chát trên 2 máy tính có kết nối với nhau). Gợi ý: viết 2 module trên 2 máy theo cơ chế client – server và kết nối với nhau qua tcp.

**YÊU CẦU CHUNG**: THỰC HIỆN THEO TRÌNH TỰ, học để hiểu, tự tìm hiểu. Bắt đầu với 1 phần mới nên học khái niệm chắc và hiểu, đưa ra được ví dụ và các so sánh giữa các phương pháp; Về phần bài tập yêu cầu comment cho code đầy đủ, gọn gàng; làm thành tài liệu chi tiết.

**Tự học thêm các khái niệm, các kỹ thuật khác không nằm trong đề cương, quy trình phát triển phần mềm (quy trinh RUP, UML, Web service …), Phần Advance tuy nói là advance nhưng bài tập mới dừng ở mức độ làm để hiểu;**

**NÊN TẠO KẾ HOẠCH CỤ THỂ LÀ TỪ NGÀY NÀO ĐẾN NGÀY NÀO LÀM GÌ? ĐẠT ĐƯỢC GÌ? HẾT 1 PHẦN (BASIC, ADVANCE) CÓ THỂ GỬI LẠI REVIEW CŨNG ĐƯỢC (BAO GỒM CẢ TÀI LIỆU VÀ BÀI TẬP)**