線程安全模式的懶漢式:

**class** Bank{

**private** Bank(){}

**private** **static** Bank *instance* = **null**;

**public** **static** Bank getInstance(){

//方式一：效率稍差

// synchronized (Bank.class) {

// if(instance == null){

// instance = new Bank();

// }

// return instance;

// }

//方式二：效率更高

**if**(*instance* == **null**){

**synchronized** (Bank.**class**) {

**if**(*instance* == **null**){

*instance* = **new** Bank();

}

}

}

**return** *instance*;

}

}

面試題:寫一個線程安全的單例模式:餓漢式、懶漢式(上面提供的)

死鎖的理解：

不同的線程分別占用對方需要的同步資源不放棄，都在等待對方放棄自己需要的同步資源，就形成了線程的死鎖

1. 說明：

1）出現死鎖後，不會出現異常，不會出現提示，只是所有的線程都處於阻塞狀態，無法繼續

2）我們使用同步時，要避免出現死鎖。

1. 舉例：

**public** **static** **void** main(String[] args) {

StringBuffer s1 = **new** StringBuffer();

StringBuffer s2 = **new** StringBuffer();

**new** Thread(){

@Override

**public** **void** run() {

**synchronized** (s1){

s1.append("a");

s2.append("1");

**try** {

Thread.*sleep*(100);

} **catch** (InterruptedException e) {

e.printStackTrace();

}

**synchronized** (s2){

s1.append("b");

s2.append("2");

System.***out***.println(s1);

System.***out***.println(s2);

}

}

}

}.start();

**new** Thread(**new** Runnable() {

@Override

**public** **void** run() {

**synchronized** (s2){

s1.append("c");

s2.append("3");

**try** {

Thread.*sleep*(100);

} **catch** (InterruptedException e) {

e.printStackTrace();

}

**synchronized** (s1){

s1.append("d");

s2.append("4");

System.***out***.println(s1);

System.***out***.println(s2);

}

}

}

}).start();

}