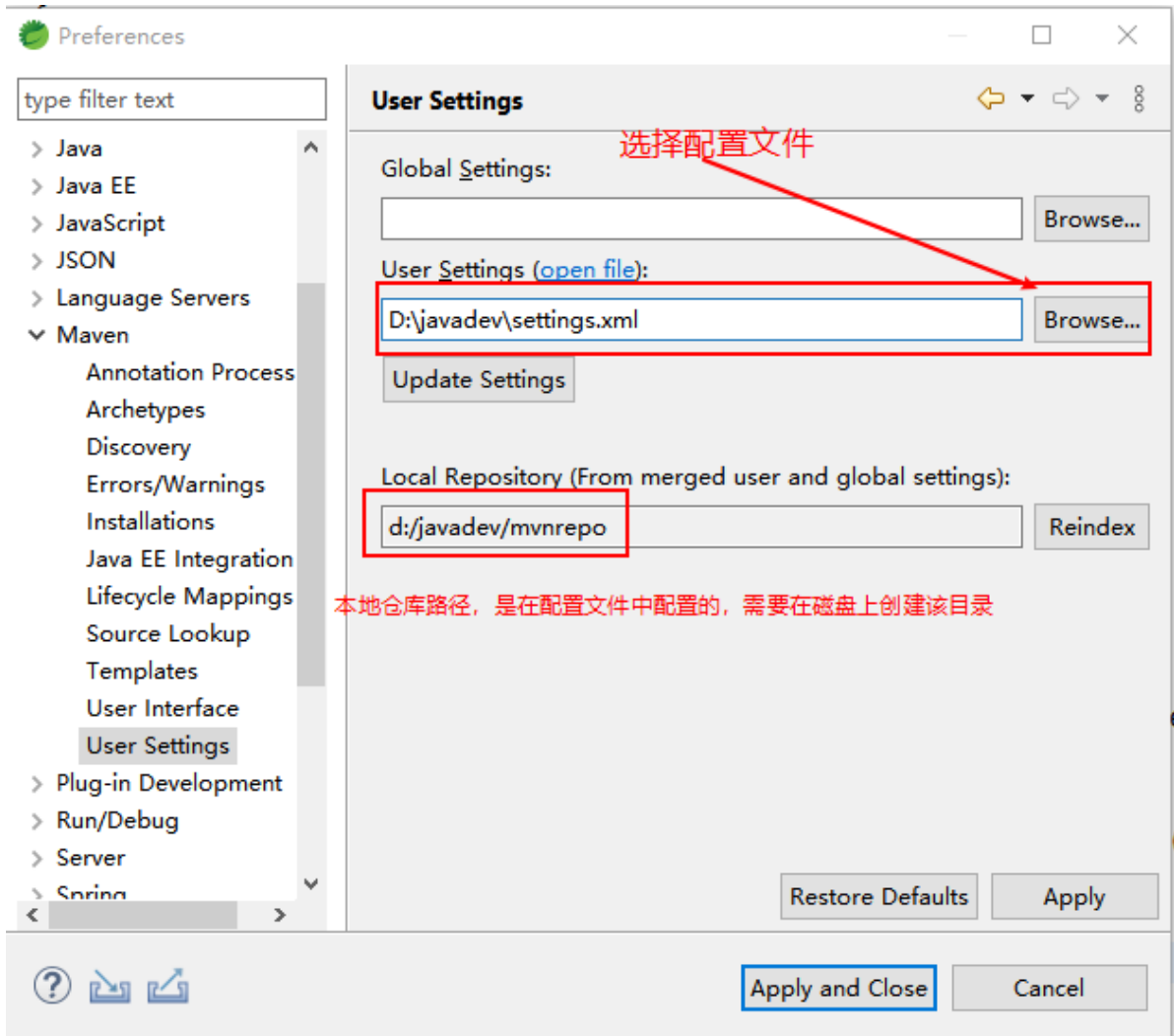


Maven配置以及项目创建

1. 选择配置文件



2. 修改配置文件内容

```
<!-- Default: ${user.home}/.m2/repository -->
<localRepository>/path/to/local/repo</localRepository>
-->
<localRepository>d:/javadev/mvnrepo</localRepository>
<!-- interactiveMode -->
| This will determine whether maven prompts you when it needs input
| maven will use a sensible default value, perhaps based on some ot
```

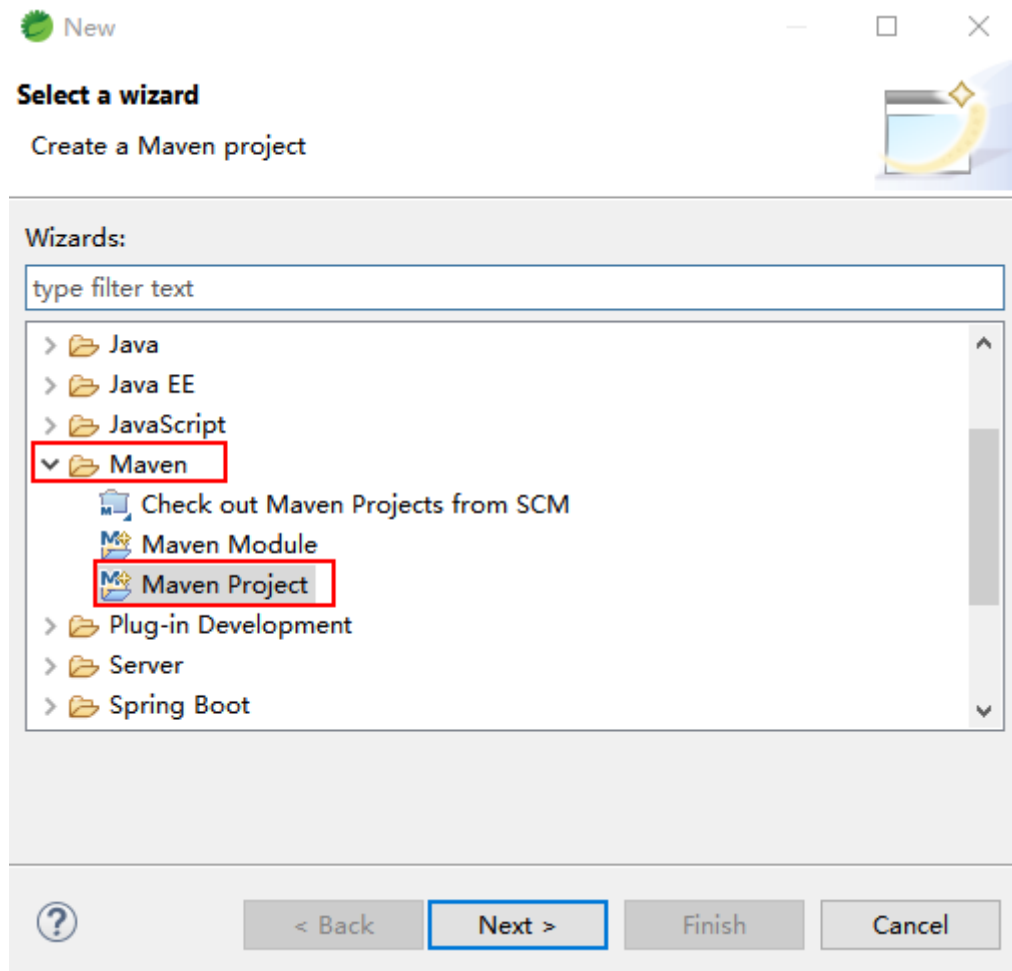
```
177<mirror>
178  <id>nexus-aliyun</id>
179  <mirrorOf>*</mirrorOf>
180  <name>Nexus aliyun</name>
181  <url>http://maven.aliyun.com/nexus/content/groups/public</url>
182</mirror>
183
```

```
<profile>
  <id>jdk-1.8</id>
  <activation>
    <activeByDefault>true</activeByDefault>
    <jdk>1.8</jdk>
  </activation>
  <properties>
    <maven.compiler.source>1.8</maven.compiler.source>
    <maven.compiler.target>1.8</maven.compiler.target>
    <maven.compiler.compilerVersion>1.8</maven.compiler.compilerVersion>
  </properties>
</profile>
```

配置JDK的版本

创建Maven项目

第一步:



第二步:

New Maven project

Select an Archetype



如果默认分类没有出现下面的列表，这里选择Internal

Catalog: Internal Configure...

Filter:

Group Id	Artifact Id	Version
org.apache.maven.archetypes	maven-archetype-archetype	1.0
org.apache.maven.archetypes	maven-archetype-j2ee-simple	1.0
org.apache.maven.archetypes	maven-archetype-plugin	1.2
org.apache.maven.archetypes	maven-archetype-plugin-site	1.1
org.apache.maven.archetypes	maven-archetype-portlet	1.0.1
org.apache.maven.archetypes	maven-archetype-profiles	1.0-alpha-4
org.apache.maven.archetypes	maven-archetype-quickstart	1.1

An archetype which contains a sample Maven project.

☒ Show the last version of Archetype only ☐ Include snapshot archetypes Add Archetype...

► Advanced

? < Back Next > Finish Cancel


选择创建quickstart类型项目

第三步

New Maven Project

New Maven project

Specify Archetype parameters



Group Id: 包名

Artifact Id: 项目名称

Version:

Package:


Properties available from archetype:

Name	Value

Add...

Remove

Advanced



< Back

Next >

Finish

Cancel

POI包的使用

- 首先导入包依赖。

在pom.xml中添加一段依赖代码

```
<dependency>
  <groupId>org.apache.poi</groupId>
  <artifactId>poi-ooxml</artifactId>
  <version>4.1.2</version>
</dependency>
```

- 写一个读Excel的例子

创建excel文件score.xlsx如下:

[illegible]

编写代码

```
package com.hqyj.javaadvanceday06;

import java.io.IOException;

import org.apache.poi.ss.usermodel.CellType;
import org.apache.poi.xssf.usermodel.XSSFWorkbook;
import org.apache.poi.xssf.usermodel.XSSFRow;
import org.apache.poi.xssf.usermodel.XSSFSheet;
import org.apache.poi.xssf.usermodel.XSSFWorkbook;

public class ReadExcel {

    public static void main(String[] args) {
        XSSFWorkbook workbook = null;
        try {
            //整个excel表格
            workbook = new XSSFWorkbook("d:\\score.xlsx");
            //获得一个sheet
            XSSFSheet sheet = workbook.getSheet("sheet1");

            int rowNum = sheet.getLastRowNum();
            for(int i=0; i<=rowNum; i++) {
                //行
                XSSFRow row = sheet.getRow(i);
                short cellNum = row.getLastCellNum();
                for(int j=0; j<cellNum; j++) {
                    //单元格
                    XSSFWorkbook cell = row.getCell(j);
                    CellType type = cell.getCellType();
                    switch(type) {
                        case NUMERIC:
                            System.out.print("\t\t" + cell.getNumericCellValue());
                            break;
                        case STRING:
                            System.out.print("\t\t" + cell.getStringCellValue());
                            break;
                    }
                }
            }
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```



```
@Test
public void testAdd() {
    assertEquals("", 5, mathDemo.add(3, 2)); //断言
}

@Test
public void testMulti() {
    assertEquals("", 15, mathDemo.multi(3, 5)); //断言
}
}
```

多线程

进程与线程

每个程序运行都会产生一个进程，操作系统会为每个进程分配资源，启动一个进程开销比较大

线程是进程中多任务，可以同时执行，由进程来管理，开销相比进程小

同步与异步

单线程程序，所有的语句排队以此执行，称为同步执行

多线程程序，同时执行多个任务，相互之间不需要等待，称为异步

线程安全与线程不安全

线程不安全指多线程异步执行时，多个线程同时访问共享资源

线程安全指让程序同步（排队）执行

Java创建多线程

继承Thread

继承Thread，重新run方法，启动线程方式是调用start()方法

```
package com.hqyj.javaadvanceday06;

public class ThreadDemo extends Thread{

    public ThreadDemo(String name) {
        super(name);
    }

    @Override
    public void run() {
        for(int i=0; i<100; i++) {
            System.out.println(getName() + " " + i);
        }
    }
}
```

实现Runnable接口

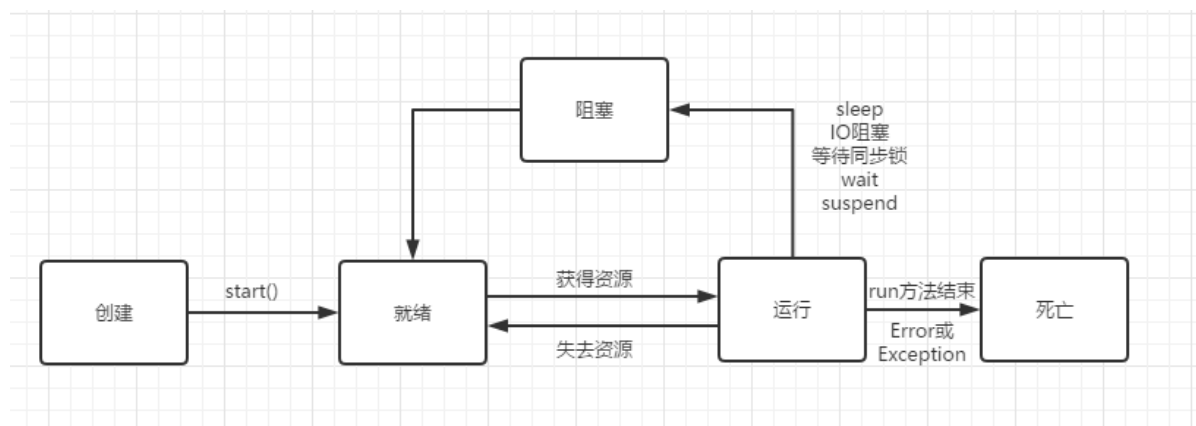
启动线程方式: `new Thread(new RunnableDemo()).start()`

```
package com.hqyj.javaadvanceday06;

public class RunnableDemo implements Runnable{

    @Override
    public void run() {
        for(int i=0; i<100; i++){
            //Thread.currentThread().getName()获取当前线程的名字, 默认名字Thread-0,
            Thread-1...
            System.out.println(Thread.currentThread().getName() + ": " + i);
        }
    }
}
```

线程的生命周期 (状态)



线程同步 (解决线程不安全问题)

银行取钱案例, 如果多线程取钱, 可能会出现余额变负数。

解决这个线程不安全问题, 把取钱的代码块加上同步关键字synchronized

第一步: 创建账号类Account

```
package com.hqyj.javaadvanceday06;

public class Account {
    private String accountNo;
    private double blance;

    public Account(String accountNo, double blance) {
        this.accountNo = accountNo;
        this.blance = blance;
    }

    public String getAccountNo() {
```



```

        return accountNo;
    }
    public void setAccountNo(String accountNo) {
        this.accountNo = accountNo;
    }
    public double getBlance() {
        return blance;
    }
    public void setBlance(double blance) {
        this.blance = blance;
    }
}

```

第二步：创建多线程取钱类DrawThread

```

package com.hqyj.javaadvanceday06;

public class DrawThread extends Thread{
    private Account account;
    private double drawAmount;

    public DrawThread(Account account, double drawAmount) {
        this.account = account;
        this.drawAmount = drawAmount;
    }

    @Override
    public void run() {
        //synchronized同步关键字，括起来的代码会同步执行
        //小括号里面这个对象称为同步器/同步锁
        synchronized (account) {
            //取钱金额小于等于账户余额，执行取钱动作
            if(drawAmount<=account.getBlance()) {
                System.out.println(getName() + "取钱成功，取了：" + drawAmount);
                account.setBlance(account.getBlance()-drawAmount);//修改余额
                System.out.println("账户余额是：" + account.getBlance());
            }else { //余额不足，不能取钱
                System.out.println("余额不足,取钱失败");
            }
        }
    }
}

```

第三步：测试两个线程取钱。

当没有添加同步关键字时，很容易出现余额为负数的情况。当添加上同步关键字时，就不会出现问题

```

Account account = new Account("894653413478967687", 888);
DrawThread thread1 = new DrawThread(account, 500);
thread1.start();
DrawThread thread2 = new DrawThread(account, 500);
thread2.start();

```

死锁

形成死锁四个条件

- 1.互斥使用。一个资源被某个线程占用时，其它线程就不能使用
- 2.不可抢占。不能强行从别的线程夺取人家占用资源，只能等释放
- 3.请求和保持。我在请求其它资源的时候，自己占用的资源不会释放
- 4.循环等待。t1线程占有A资源，需要等待B资源；t2线程占有B，需要等待A资源

死锁代码举例

```
package com.hqyj.javaadvanceday06;

public class DeadLock extends Thread{
    A a = new A();
    B b = new B();

    @Override
    public void run() {
        a.foo(b);
    }

    public void other() {
        b.foo(a);
    }

    public static void main(String[] args) {
        DeadLock lock = new DeadLock();
        lock.start();
        lock.other();
    }
}

class A{
    public synchronized void foo(B b) {
        System.out.println("A调用B的bar()");
        b.bar();
    }

    public synchronized void bar() {
        System.out.println("执行a.bar()");
    }
}

class B{
    public synchronized void foo(A a) {
        System.out.println("B调用A的bar()");
        a.bar();
    }
}
```

```
    public synchronized void bar() {  
        System.out.println("执行b.bar()");  
    }  
}
```

线程池

```
package com.hqyj.javaadvanceday06;  
  
import java.util.concurrent.ExecutorService;  
import java.util.concurrent.Executors;  
  
public class ThreadPoolDemo extends Thread{  
  
    @Override  
    public void run() {  
        for(int i=0; i<20; i++) {  
            System.out.println(getName() + ": " + i);  
        }  
    }  
  
    public static void main(String[] args) {  
        ExecutorService pool = Executors.newCachedThreadPool(); //创建线程池对象  
        pool.submit(new ThreadPoolDemo()); //把线程放到线程池，线程启动由线程池来管理的  
        pool.submit(new ThreadPoolDemo());  
    }  
}
```

今日作业

- 1.进程是什么？线程是什么？它们之间有何关系？
- 2.同步和异步有什么异同？
- 3.请举例描述线程不安全？
- 4.简述线程的状态，以及状态之间的转换？
- 5.死锁形成的条件？