JavaSE复习

线程

概念

线程（thread）：比进程更小的运行单位，是程序中单个顺序的流控制，一个进程中可以包含多个线程。

作用

1. 充分利用cpu资源，减少cpu空闲时间，让cpu执行更多的任务，加快程序运行，提高效率。

程序、进程、多任务和线程的区别

1. 程序（program）是对数据描述与操作的代码的集合，是应用程序执行的脚本。
2. 进程（process）是程序的一次执行过程，是系统运行程序的基本单位。程序是静态的，进程是动态的。系统运行一个程序即是一个进程从创建、运行到消亡的过程。
3. 多任务（multi task）在一个系统中可以同时运行多个程序，即有多个独立运行的任务，每个任务对应至少一个进程。
4. 线程（thread）：比进程更小的运行单位，是程序中单个顺序的流控制。一个进程中可以包含多个线程。

线程创建方式

1. 继承Thread类，重写run方法
    - 继承Thread的类不能再继承其他类
2. 实现Runnable接口，重写run方法
    - 实现Runnable接口的类可以多实现
    - 实现Runnable接口的方式更容易实现线程共享
    - 实现Runnable接口的类不能直接运行，需要通过Thread类运行

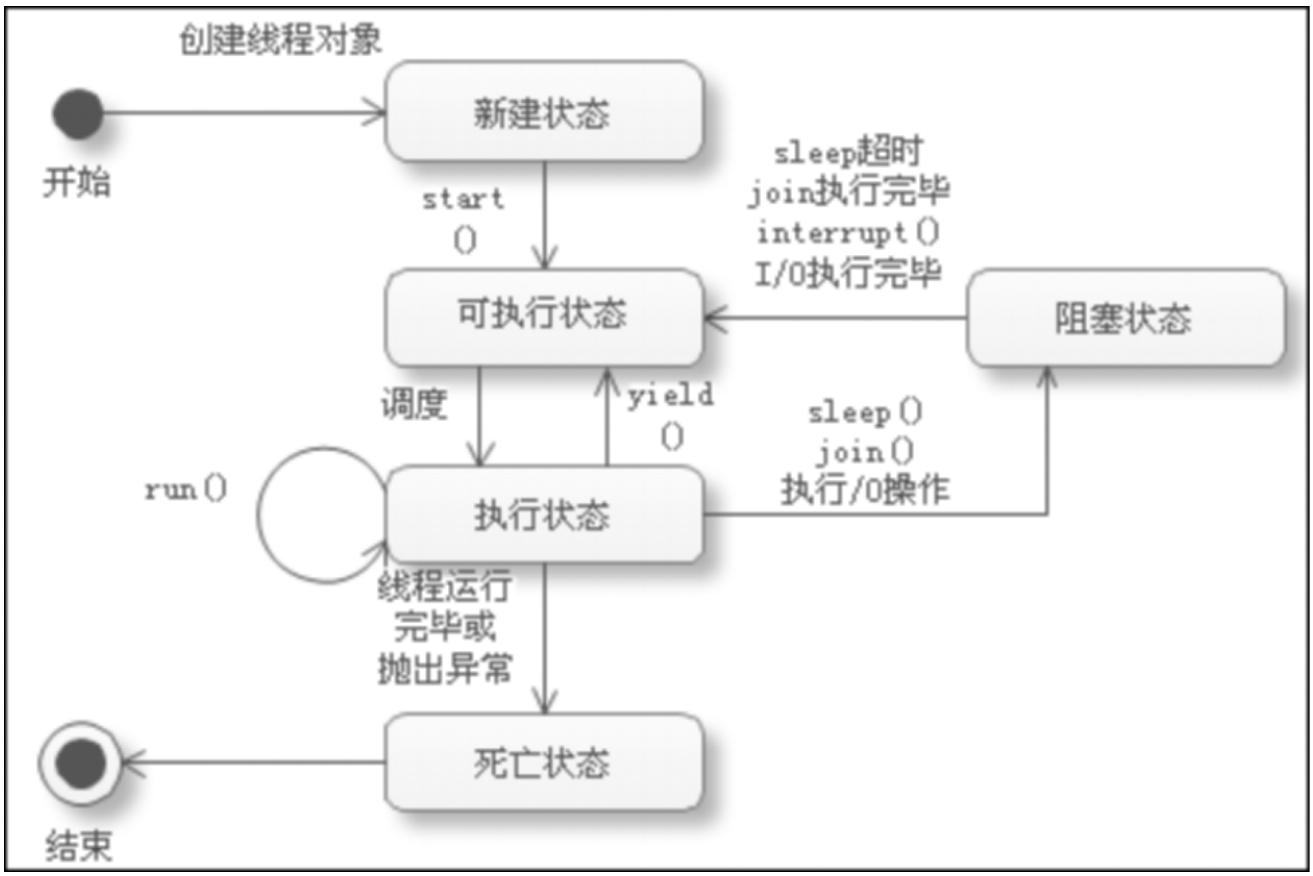
```
public class MyRunnable implements Runnable {
    public void run() {
        ...
    }
}

//运行
MyRunnable runnable = new MyRunnable();
new Thread(runnable).start();
```

3. 通过线程池Excutors创建线程，实现线程的重用，减少资源消耗。
    - Excutors.newCachedThreadPool()， 缓存线程池
    - Excutors.newFixedThreadPool(int nThreads)， 固定数目线程池

- Excutors.newScheduledThreadPool(int corePoolSize)，调度线程池
- Excutors.newWorkStealingPool()，工作窃取线程池

线程的生命周期

1. 创建状态（新建状态）
2. 就绪状态（可执行状态）
3. 运行状态（执行状态）
4. 阻塞状态
5. 死亡状态



线程的方法

1. run方法：当前线程获得cpu控制权就会调用该方法
2. start方法：线程创建后执行该方法启动线程，让线程进入可执行状态状态，等待cpu调度
3. yield方法：处于执行状态的线程调用该方法会让出cpu控制权，回到可执行状态
4. join方法：处在“执行状态”的线程如果调用了其他线程的 join 方法，当前线程被挂起进入“阻塞状态”
5. sleep方法：处在“执行状态”的线程调用该方法会休眠一段时间，**不让出cpu控制权**，所以该线程进入阻塞状态
6. wait方法：处在“执行状态”的线程调用该方法会中断执行，使本线程等待，暂时**让出 cpu 的使用权**，并允许其他线程使用这个同步方法
7. notify/notifyAll方法，notify唤醒等待的线程，notifyAll唤醒所有等待的线程，结合wait方法使用

线程的调度

1. 分时调度模型
2. **抢占式调度模型**
    - 线程类设置了10个优先级，分别使用1~10内的整数表示，整数值越大代表优先级越高。每个线程都有一个默认的优先级，主线程的默认优先级是5

线程的同步

原因

多线程应用程序同时访问共享对象时，由于线程间相互抢占CPU的控制权，造成一个线程夹在另一个线程的执行过程中运行,所以可能导致错误的执行结果。

解决方案

为了防止共享对象在并发访问时出现错误，Java中提供了“synchronized”关键字。**synchronized关键字**确保共享对象在同一时刻只能被一个线程访问，这种处理机制称为“线程同步”或“线程互斥”。Java中的**“线程同步”基于“对象锁”**的概念。

1. 同步方法：被“synchronized”关键字修饰的方法称为”同步方法”。

```
//定义同步方法
public synchronized void methd(){
   //方法实现
}
```

2. 同步代码块

```
synchronized(同步对象){
    同步操作
}
```

线程通信

当一个线程使用的同步方法中用到某个变量，而此变量又需要其他线程修改后才能符合本线程的需要，那么可以在同步方法中使用 wait() 方法等待其他线程唤醒。

wait()方法:中断方法的执行，使本线程等待，暂时让出 cpu 的使用权，并允许其他线程使用这个同步方法。

notify()方法：唤醒由于使用这个同步方法而处于等待线程的某一个线程结束等待

notifyall()方法：唤醒所有由于使用这个同步方法而处于等待的线程结束等待

sleep和wait区别

1. sleep：睡眠，它是Thread方法，阻塞，不会让出cpu 的使用权，不允许其他线程使用这个同步方法
2. wait：等待，它是Object方法，不会阻塞，让出cpu 的使用权，允许其他线程使用这个同步方法，后续需要notify或者notifyall唤醒

死锁

> 两个线程互相等待对方运行结果就会造成死锁。

解决方案：

1. 让线程有序执行，先执行谁再执行谁
2. 设置线程超时时间，超时时间过后重试或者回退

网络编程

概念

http、https、tcp、udp区别：

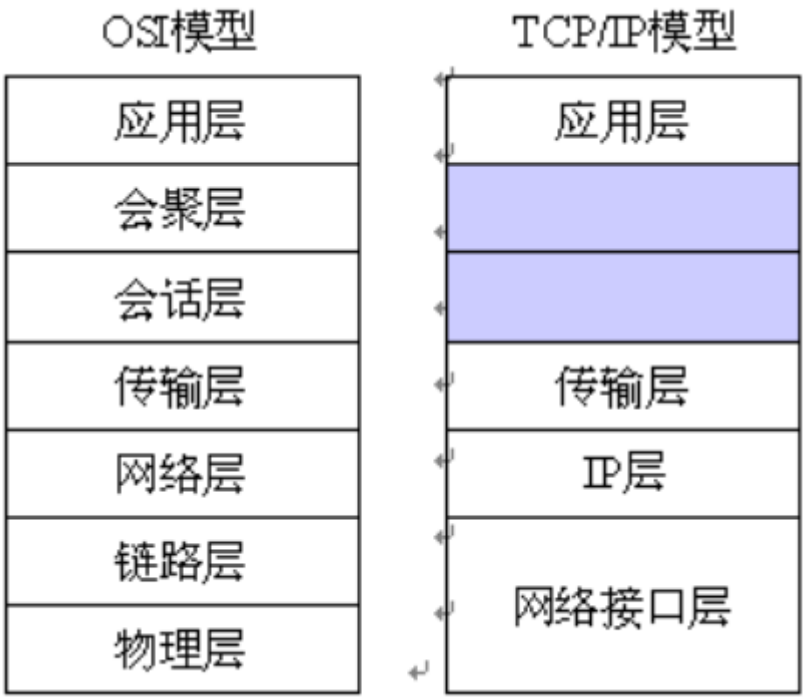
http：应用层的传输协议，无状态短连接

https：应用层的传输协议，无状态短连接，传输内容进行加密，安全可靠

tcp：传输层的协议，有状态长连接，http、https是建立在tcp协议上的，保证内容一定达到目标主机，实时性低但安全可靠，一般用在交易、一般网络的应用。“三次握手，四次挥手”。

udp：传输层协议，有状态长连接，不需要提前建立网络通道，不保证传输内容一定达到目标主机，传输速度快，实时性高，一般应用在聊天、直播等

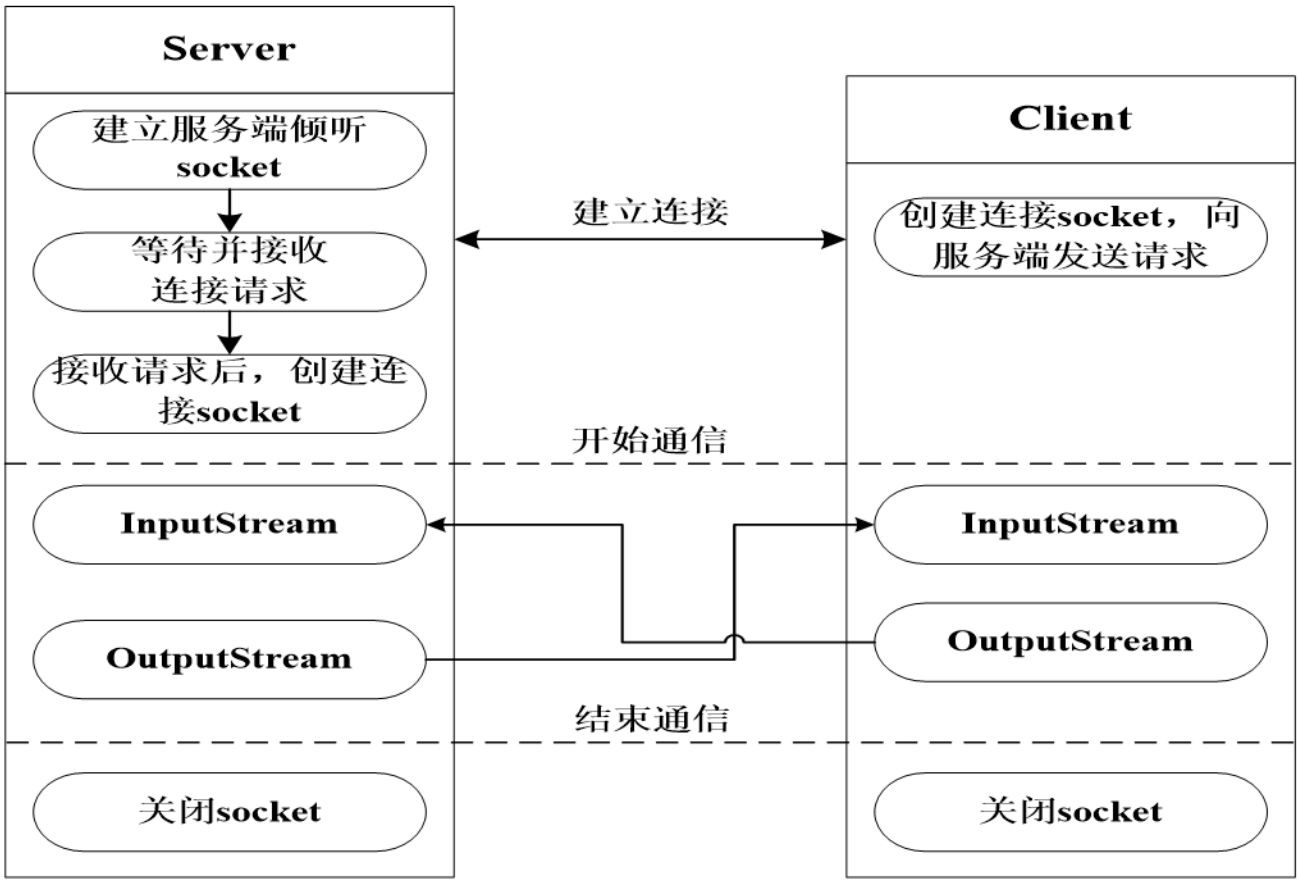
网络架构



Socket

一个双向的网络通信连接实现数据交换，这个双向链路的一段称为一个Socket (套接字)。

tcp（c/s）



客户端Socket的工作过程包含以下四个基本的步骤：

1. 创建 Socket。根据指定的 IP 地址或端口号构造 Socket 类对象。如服务器端响应，则建立客户端到服务器的通信线路。
2. 打开连接到 Socket 的输入/出流。 使用 getInputStream（）方法获得输入流，使用 getOutputStream（）方法获得输出流。
3. 按照一定的协议对 Socket 进行读/写操作。通过输入流读取服务器放入线路的信息（但不能读取自己放入线路的信息），通过输出流将信息写入线程。
4. 关闭 Socket。断开客户端到服务器的连接，释放线路

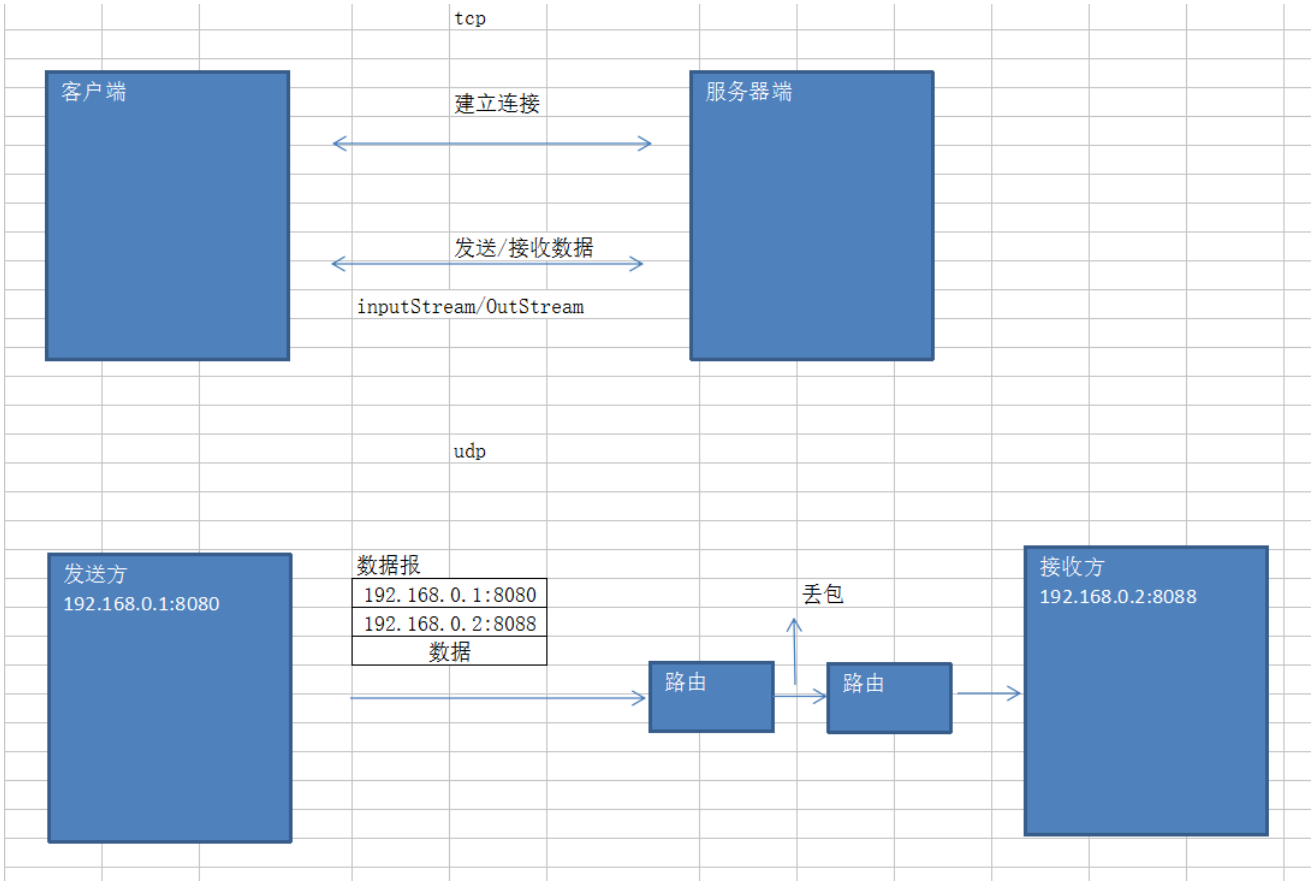
服务器程序的工作过程包含以下四个基本的步骤：

对于服务器和客户机而言，将上述第一步改为构造 ServerSocket 类对象，监听客户端的请求并进行响应。

1. 调用 ServerSocket(int port) 创建一个服务器端套接字，并绑定到指定端口上。
2. 调用 accept()，监听连接请求，如果客户端请求连接，则接受连接，返回通信套接字。
3. 调用 Socket类的 getOutputStream 和 getInputStream 获取输出流和输入流，开始网络数据的发送和接收。
4. 最后关闭通信套接字。

udp

类 DatagramSocket 和 DatagramPacket 实现了基于 UDP 协议网络程序。DatagramPacket 对象封装了UDP数据报，在数据报中包含了客户端的IP地址和端口号以及服务器的IP地址和端口号。UDP数据报通过数据报套接字 DatagramSocket 发送和接收，系统不保证UDP数据报一定能够安全送到目的地，也不能确定什么时候可以抵达。



类加载机制

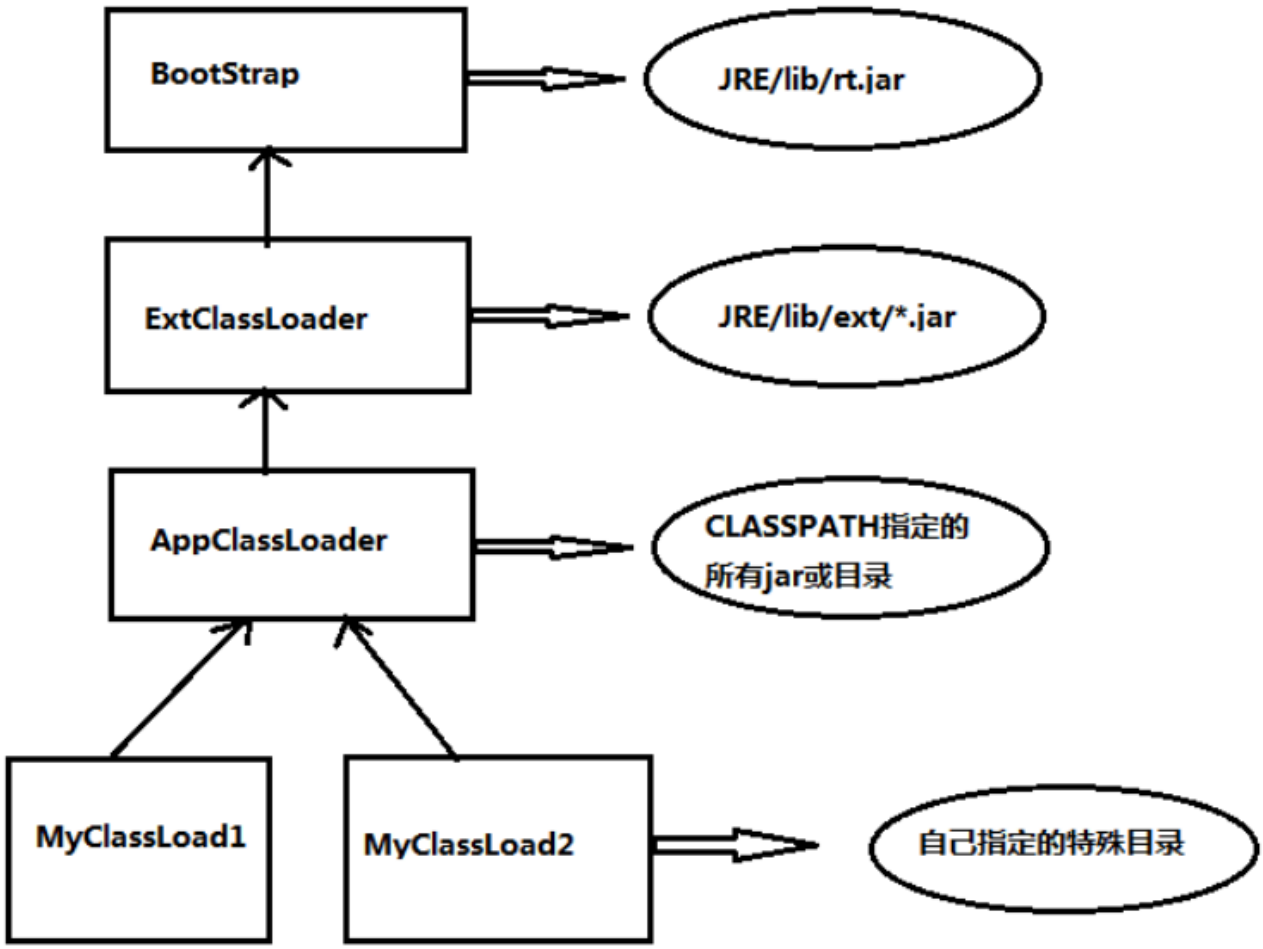
什么是类加载

类的加载指的是将类的.class文件中的二进制数据读入内存中，将其放在运行时数据区域的方法去内，然后在堆中创建java.lang.Class对象，用来封装类在方法区的数据结构。只有java虚拟机才会创建class对象，并且是一一对应关系。这样才能通过反射找到相应的类信息。

什么是类加载器

负责对类的加载的组件叫类加载器。

1. 根类加载器,使用c++编写(BootStrap),负责加载rt.jar
2. 扩展类加载器,java实现(ExtClassLoader)
3. 应用加载器,java实现(AppClassLoader) classpath

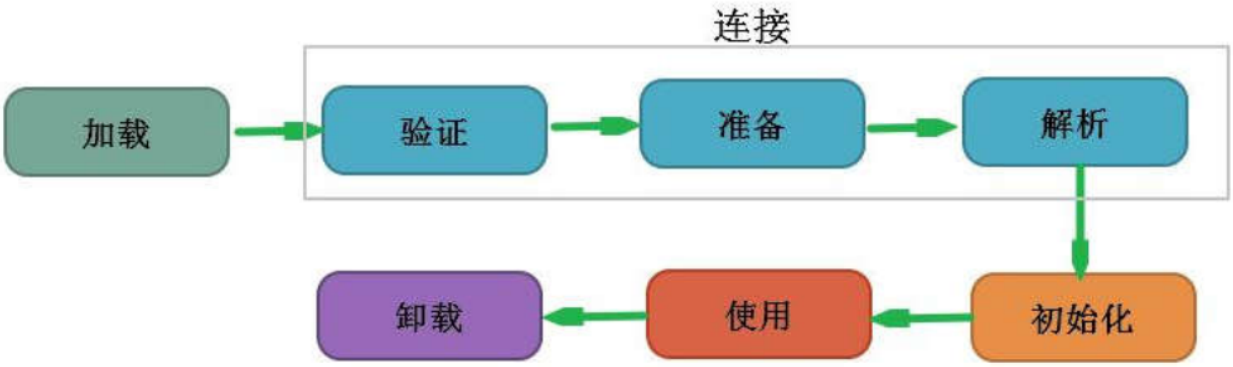


双亲委派机制

比如A类的加载器是AppClassLoader(其实我们自己写的类的加载器都是AppClassLoader)，AppClassLoader不会自己去加载类，而会委ExtClassLoader进行加载，那么到了ExtClassLoader类加载器的时候，它也不会自己去加载，而是委托BootStrap类加载器进行加载，就这样一层一层往上委托，如果Bootstrap类加载器无法进行加载的话，再一层层往下走。

类加载过程

JVM类加载过程分为五部分：加载、验证、准备、解析、初始化。



类的生命周期

加载

将类的class文件读入内存中，并创建该类Class对象。

连接

1. 验证

   确保class文件的字节流中包含的信息符合当前虚拟机的要求，并且不会危害虚拟机的安全。

2. 准备

   为类变量（static修饰的变量）分配内存，并设置类变量的初始值。

3. 解析

   将常量池的符号引用替换成直接引用（内存地址）。

初始化

执行class文件的代码，按照一定规则执行：

(1)创建对象的实例：我们new对象的时候，会引发类的初始化，前提是这个类没有被初始化。

(2)调用类的静态属性或者为静态属性赋值

(3)调用类的静态方法

(4)通过class文件反射创建对象

(5)初始化一个类的子类：使用子类的时候先初始化父类

(6)java虚拟机启动时被标记为启动类的类：就是我们的main方法所在的类