1 代码分析

- 父/子类常量->父类静态属性->子类静态属性->父类实例属性->父类构造器->子类实例属性->子类构造器
- 属性就是字段，**不是方法**，但是字段也可以由方法来赋值，这样就可以通过初始化属性来间接调用方法

```
public class 父类 {
    private String str = getStr();
    {
        System.out.println("父类的实例块");
    }
    public 父类() {
        System.out.println("父类的构造方法");
    }
    private static String staticStr = getStaticStr();//运行到这里时， CONST == "父类staticfinal"，CONS == null：说明 static final 修饰的常量，在类初始化之前就已赋值(准确的是在准备阶段)，修饰的方法和普通的 static 修饰赋值时间一样！
    //测试 static final 修饰的变量赋值时间
    public static final String CONST = "父类staticfinal";
    public static final String CONS = staticFinalSup();
    static {
        System.out.println("父类的静态块");
    }
    private static String getStaticStr() {
        System.out.println("父类的静态属性初始化");
        return null;
    }
    private String getStr() {
        System.out.println("父类的实例属性初始化");
        return null;
    }
    private static String staticFinalSup() {
        System.out.println("父类 static final 方法");
        return "父类 Static Final";
    }
}
public class 子类 extends 父类{

    {
        System.out.println("子类的实例块");
    }
    public 子类() {
        System.out.println("子类的构造方法");
    }
```

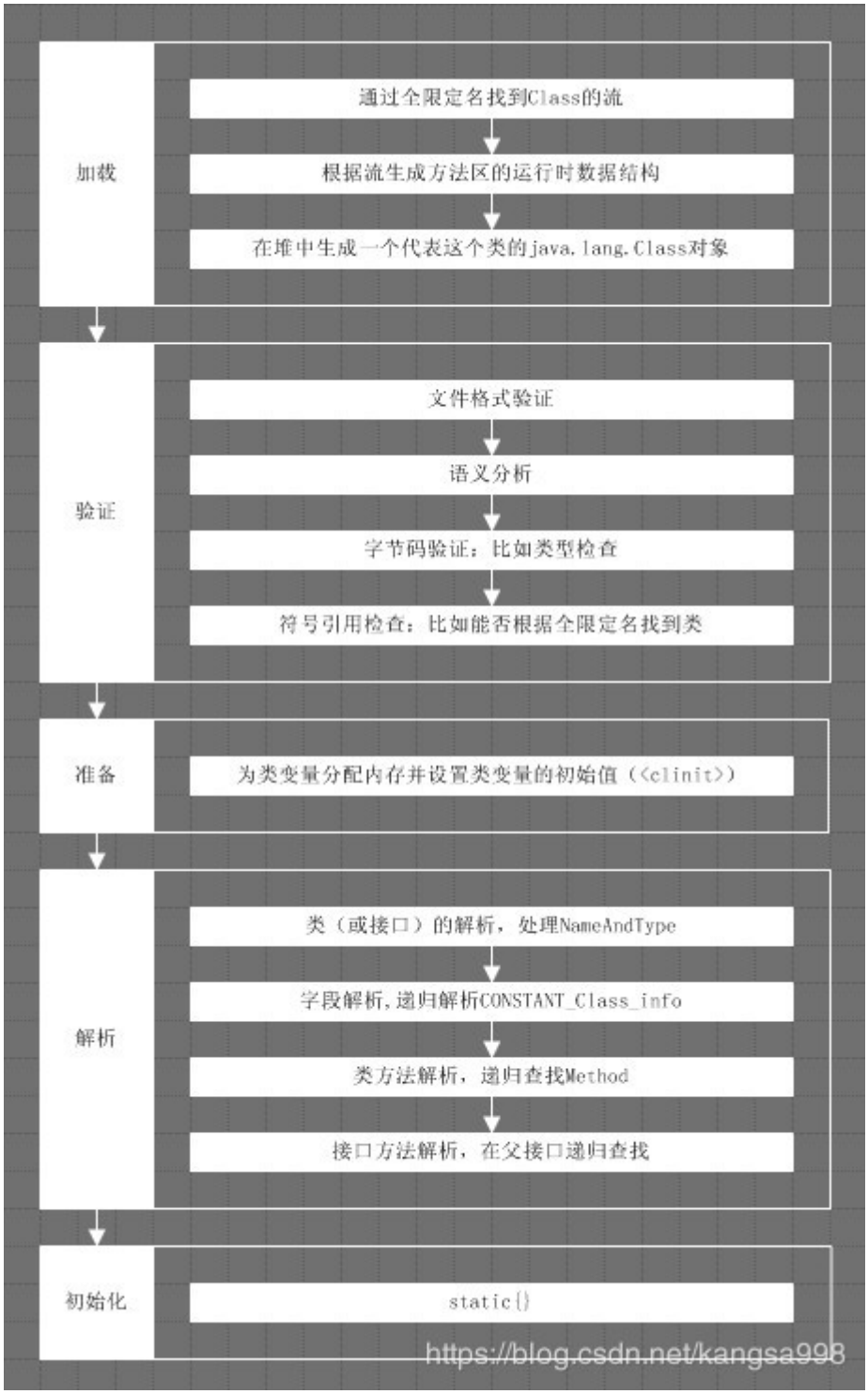
```
    static {
        System.out.println("子类的静态块");//运行到这里时，CON == String，CONST == null，也说明了父类一样的问题
    }
    //测试 static final 什么时候赋值
    public static final String CONST = staticFinalSub();
    public static final String CON = "String";
    private static String staticStr = getStaticStr();
    private String str = getStr();
    private static String getStaticStr() {
        System.out.println("子类的静态属性初始化");
        return null;
    }
    private String getStr() {
        System.out.println("子类的实例属性初始化");
        return null;
    }
    public static void main(String[] args) {
        new 子类();
        System.out.println("--------------------");
        new 子类();
    }
    public static String staticFinalSub() {
        System.out.println("子类 static final 方法");
        return "子类 Static Final";
    }
}
```

```
父类的静态属性初始化
父类 static final 方法
父类的静态块

子类的静态块
子类 static final 方法
子类的静态属性初始化
-------------------------//以上可以说明，首先执行父类静态属性，子类静态属性，且静态属性的执行顺序和声明顺序一样。并且 static final 修饰的方法和普通的 static 修饰的方法的执行时间是同一类

父类的实例属性初始化
父类的实例块
父类的构造方法
-------------------------//以上可以说明，执行完静态属性后，执行父类实例属性（顺序也是声明顺序），最后执行构造方法

子类的实例块
子类的实例属性初始化
子类的构造方法
-------------------------//以上可以说明，最后执行 子类实例属性，子类构造方法

--------------------
父类的实例属性初始化
父类的实例块
父类的构造方法
子类的实例块
```

子类的实例属性初始化
子类的构造方法
-------------------------//以上可以说明，静态属性只执行一次，而实例属性，每次初始化都要执行一次！

2 JVM 类初始化过程图

- 图片出自：java中类加载顺序（深入Java）
- 常量在准备阶段就赋值了，静态变量只是分配内存



3 6种使类初始化的方式

- 调用/赋值静态变量（注：不是常量，调用常量不会初始化）
- 调用静态方法(包括 static final 修饰的静态方法)
- new
- 反射(Class.forName(xxx))
- 初始化子类，则先初始化其父类
- 运行 main，则初始化其所在的类

参考

Java---类加载机制 java中类加载顺序（深入Java） 深入理解Java虚拟机