了解完class文件的存储数据结构以后，在class文件中描述的各种信息，都需要加载到虚拟机内存之后才能进行运行和使用。虚拟机需要把描述类的数据和文件从class文件加载到内存中，并对数据进行校验、转换、解析和初始化，最终形成可以被虚拟机直接使用的java类型，这个就是虚拟机的类加载机制

类加载的生命周期：
1. 加载
2. 验证 【连接】
3. 准备 【连接】
4. 解析 【连接】
5. 初始化
6. 使用
7. 卸载

其中，加载、验证、准备、初始化和卸载这5个阶段是确定的，类的加载过程必须按照这种顺序按部就班的开始，而解析阶段就不一定：在某些情况下可以在初始化阶段后再开始，为了支持java语言的运行时绑定【接口对应实现类】

什么时候开始类加载过程的第一个阶段加载呢？虚拟机没有进行强制约束，可以由虚拟机的具体实现自由把握。但是对于初始化阶段，虚拟机规范则是进行了严格的规范，有且只有5中情况才能对类进行初始化，这样的话加载，验证，准备自然而言要再初始化之前了

1. 遇到new、getstatic、putstatic、invokestatic这4条指令的时候，如果类没有进行初始化，则需要先触发初始化。这四种场景最常见就是：使用new关键字进行初始化对象的时候、设置和读取一个类的静态字段（被final修饰的字段，已经在编译期把结果存放到常亮池中的 除外）的时候、以及调用一个类的静态方法
2. 使用java.lang.refelect包的方法对类进行反射调用的时候，如果类没有进行初始化，则需要先触发其初始化
3. 当初始化一个类的时候，如果发现其父类还没有进行初始化，则需要先触发其父类的初始化
4. 当虚拟机启动的时候，用户需要指定一个要执行的主类，虚拟机要先初始化这个类
5. 如果使用jdk1.7的动态语言支持是，如果一个Java.lang.invoke.MethodHandle实例最后的解析结果REF_getStatic、REF_putStatic、REF_invokeStatic的方法

句柄，并且这个方法句柄所对应类没有初始化，则先触发其初始化
以上5中场景，是虚拟机中有且只有的5中方法，这五种方法被称为主动引用。除此之外，所有类的方法都不会触发初始化，称为被动引用

如下示例不会主动初始化：

```
/**
 * 如下方式不会触发主动初始化
 *
 * @author asheng
 * @since 2019/4/17
 */

class SuperClass {
    static {
        System.out.println("super class init.");
    }

    public static int value = 123;
}

class SonClass extends SuperClass {

    static {
        System.out.println("son class init.");
    }

}

/**
 * 因为子类引用父类的静态常亮，其实子类并没有任何指令，而是父类的指令，因此不会主动初始化
 * 这里的代码只会输出“super class init”而不会“son class init”
 */
class NoInitialization {
```

```
    public static void main(String[] args) {
        System.out.println(SonClass.value);
    }
}

/**
 * 如下也没有触发主动初初始化
 * 因为使用的指令是newarray，而且初始化的对象是[Superclass, 这个类的父类是object
 */
class NoInitialization1 {
    public static void main(String[] args) {
        SuperClass[] sca = new SuperClass[10];
    }
}


class ConstantClass {
    static {
        System.out.println("constant class");
    }

    public static final String HELLO = "hello";
}

/**
 * 也不会触发，因为编译的时候就会把ConstantClass.HELLO替换成"hello"
 * 然后被压入NoInitialization2的常量池中，以后就不会和ConstantClass有关联，因此也不会主动初始化
 */
class NoInitialization2 {
    public static void main(String[] args) {
        System.out.println(ConstantClass.HELLO);
```

```
    }
}
```

接口的初始化和类的基本相同，不同的是第三点，如果类中存在继承关系，也要初始化父类，而接口不一定，只有引用到接口中的成员变量的时候，才会初始化父类接口