

1. Car.class文件加载入内存【JVM】，如何加载的，加载的顺序是什么？由谁加载的？

类加载器【classload】:class文件加载到jvm内存，三个类加载器

java类【自定义】

lib\ext

lib\下的所有.class文件

BootStrap Classloader

Extenstion Classloader

Application Classloader

自定义加载器

**双亲委托机制**

Car.java -🡪Car.class-🡪Application classloader🡪?

🡪转交给ext加载器

---》转交bootstrap加载器

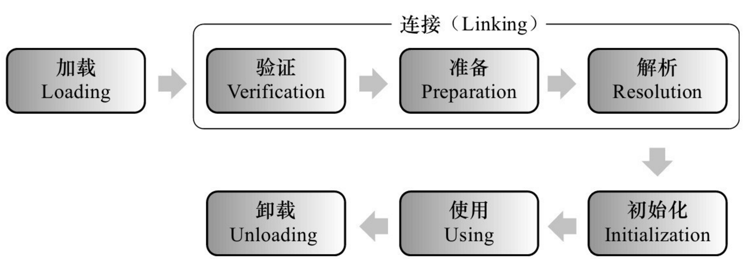
🡪验证是否能够加载

-🡪ext

ext验证

-🡪app

类加载器存在继承关系。



## 加载

根据一个类的全限定名来读取此类的二进制字节流到JVM内部;

将字节流所代表的静态存储结构转换为方法区的运行时数据结构（hotspot选择将Class对象存储在方法区中，Java虚拟机规范并没有明确要求一定要存储在方法区或堆区中）

转换为一个与目标类型对应的java.lang.Class对象；

* 通过一个类的全限定名来获取定义此类的二进制字节流（并没有指明要从一个Class文件中获取，可以从其他渠道，譬如：网络、动态生成、数据库等）；
* 将这个字节流所代表的静态存储结构转化为方法区的运行时数据结构；
* 在内存中生成一个代表这个类的java.lang.Class对象，作为方法区这个类的各种数据的访问入口；

## 连接

**验证**

验证阶段主要包括四个检验过程：文件格式验证、元数据验证、字节码验证和符号引用验证;

**准备**

为类中的所有**静态变量分配内存空间**，并为其设置一个初始值（由于还没有产生对象，实例变量将不再此操作范围内）；

**解析**

**将常量池中所有的符号引用转为直接引用**（得到类或者字段、方法在内存中的指针或者偏移量，以便直接调用该方法）。这个阶段可以在初始化之后再执行。

## 初始化

  在连接的准备阶段，类变量已赋过一次系统要求的初始值，而在初始化阶段，则是根据程序员自己写的逻辑去初始化类变量和其他资源，

* 为一个类型创建一个新的对象实例时（比如new、反射、序列化）
* 调用一个类型的静态方法时（即在字节码中执行invokestatic指令）
* 调用一个类型或接口的静态字段，或者对这些静态字段执行赋值操作时（即在字节码中，执行getstatic或者putstatic指令），不过用final修饰的静态字段除外，它被初始化为一个编译时常量表达式
* 调用JavaAPI中的反射方法时（比如调用java.lang.Class中的方法，或者java.lang.reflect包中其他类的方法）
* 初始化一个类的派生类时（Java虚拟机规范明确要求初始化一个类时，它的超类必须提前完成初始化操作，接口例外）
* JVM启动包含main方法的启动类时。

# Class对象的诞生

1. 什么时候出生？
   1. 类加载
   2. Class对象
   3. 一个类一个Class对象
   4. Class对象和类产生的实例化对象有关系
2. 取得Class对象
   1. 调用类的属性class属性



* 1. 通过实例化对象获取



1. Class对象的使用





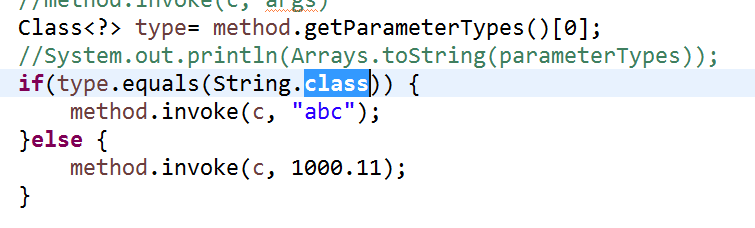
# Class对象的API

1. ---》Class对象
2. //取得类中声明的方法
3. Method
   1. getName（）；方法名称
   2. method.invoke(对象参数,); //对象.method(参数)

返回的是方法调用的返回值

* 1. method.getParameterTypes()//方法的参数类型

返回时class对象数组

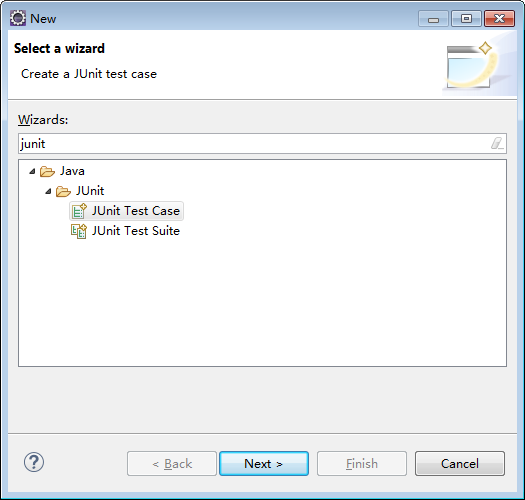


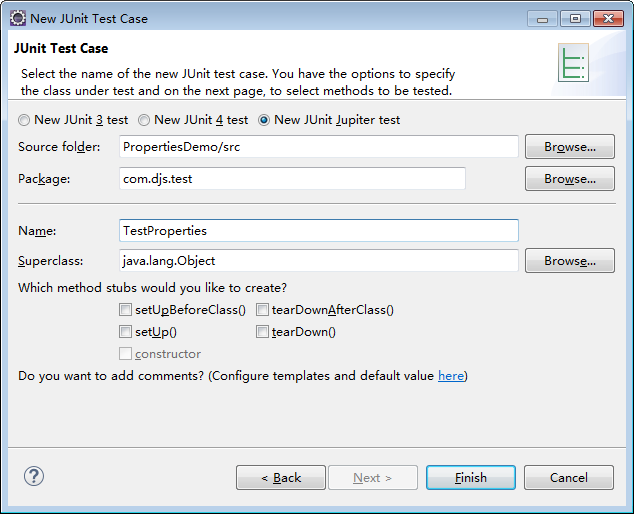
# properties文件解析

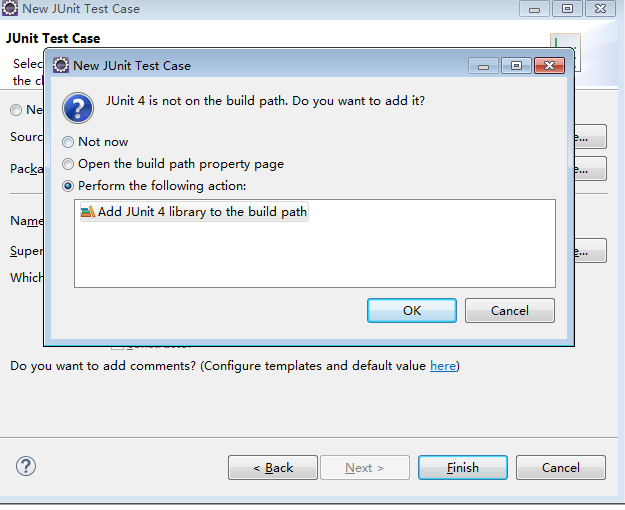
1. 配置文件的目的

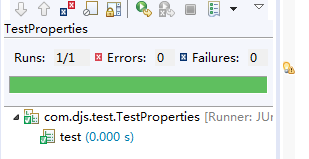
解除程序中对某些资源文件配置的耦合关系。

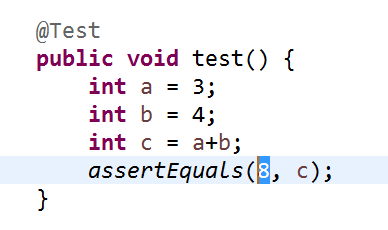
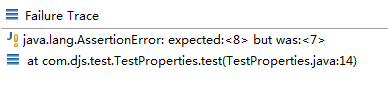
1. properties文件的基本格式
   1. 后缀： .properties
   2. 内容
      1. URL=jdbc:oracle:thin:@localhost:1521:xe
      2. USER=liuhuan
      3. PASSWORD=huan
      4. DRIVER=oracle.jdbc.OracleDriver
2. 解析方式
   1. Properties: java.util对象
   2. 类似于 Map集合m
3. Junit单元测试工具



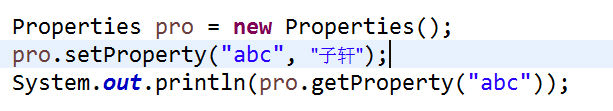










.properties文件配合使用

# XML解析

1. 什么是XML？



1. xml的目的？
   1. 存储数据
   2. ajax
   3. 文本 stu.txt,格式化【可读性比较强】
2. 格式
   1. 单标签
   2. 双标签
3. XML解析
   1. DOM

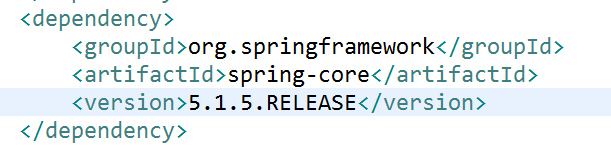
Document—》xml

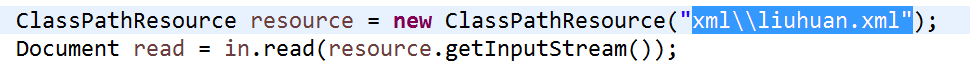
DOM加载进入内存

缺点：消耗内存比较大

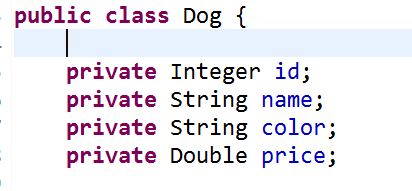
* 1. SAX
     1. 耗内存小
     2. 查询复杂

1. 路径的解决方案
   1. path---》FileInputStream
   2. Spring框架

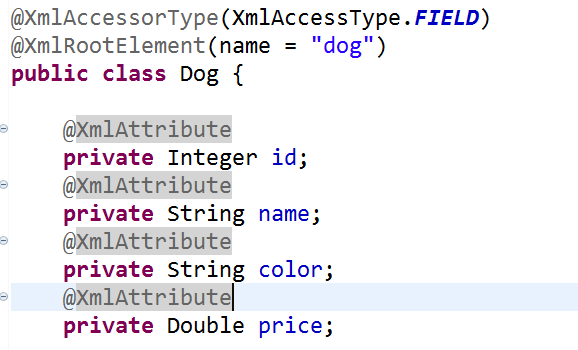


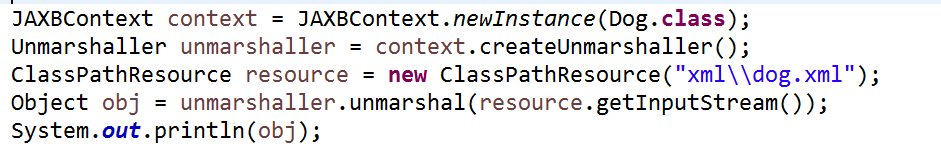


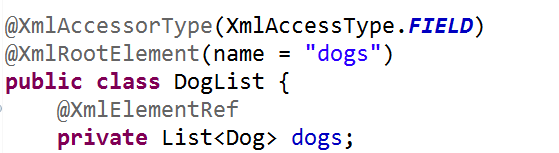
1. JABX
   1. xml文件映射为java类
   2. javax内置的











# maven创建ee项目配置

