Loading（双亲委派）

类加载器

jvm有一个类加载器的层次，分别加载不同的class

jvm中所有的class 都是被类加载器加载到内存的

一个class 被 load 到内存后， 内存中创建了两块内容

一块是将class对应的二进制内容扔进 内存中 ，同时也生成了一个class类的对象并指向二进制文件，这个对象保存在mataspace中

别的对象访问这个类的时候就是通过访问生成的这个Class类对象，然后这个class类对象再去访问二进制文件

即访问过程是 通过访问class文件对应创建的class类，然后class类从指向的二进制中获取对应二进制并解析成汇编指令

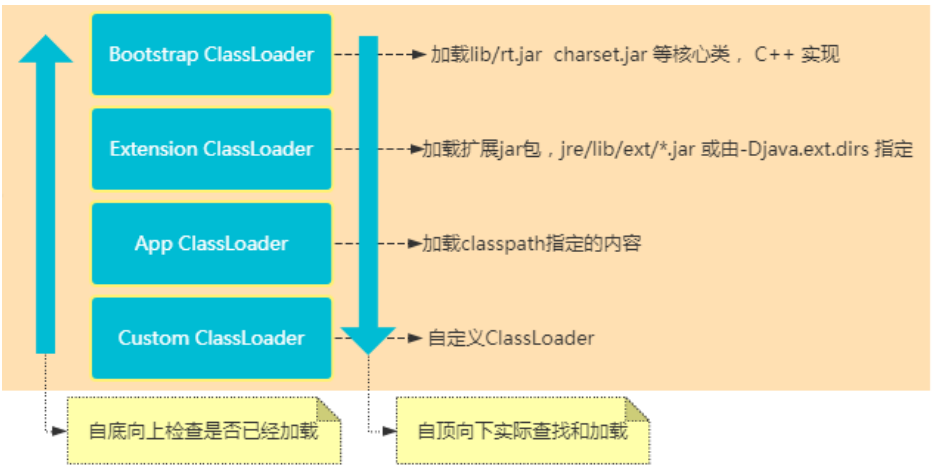
如下是获取类对应加载器的部分示例

```
public class T002_ClassLoaderLevel {
    public static void main(String[] args) {
        // null---Bootstrap ClassLoader
        // 返回null的原因：Bootstrap ClassLoader 是由C++实现的，java中没有对应的class 所以返回空
        // 可以理解成c++实现的一个模块
        System.out.println(String.class.getClassLoader());
        //null---Bootstrap ClassLoader
        System.out.println(sun.awt.HKSCS.class.getClassLoader());
        //sun.misc.Launcher$ExtClassLoader@6d6f6e28 --- Extension ClassLoader
        System.out.println(sun.net.spi.nameservice.dns.DNSNameService.class.getClassLoader());
        //sun.misc.Launcher$AppClassLoader@18b4aac2 --- App ClassLoader
        System.out.println(T002_ClassLoaderLevel.class.getClassLoader());

        // null   父加载器不是 加载器的加载器     所以   classLoader 的 classloader 都是null
        System.out.println(sun.net.spi.nameservice.dns.DNSNameService.class.getClassLoader().getClass().getClassLoader());
        // null  同上   ClassLoader 本身就是c++ 写的
        System.out.println(T002_ClassLoaderLevel.class.getClassLoader().getClass().getClassLoader());

        // sun.misc.Launcher$ExtClassLoader@6d6f6e28 --- T002_ClassLoaderLevel 的类加载器是APP APP的父加载器是 Ext加载器
        System.out.println(new T002_ClassLoaderLevel().getParent());
        System.out.println(ClassLoader.getSystemClassLoader());
    }
}
```

加载器：



Bootstrap jdk 加载核心类的类加载器

Extension 是加载扩展包中的的

App 加载classpath 指定的类的，即加载我们自己写的类

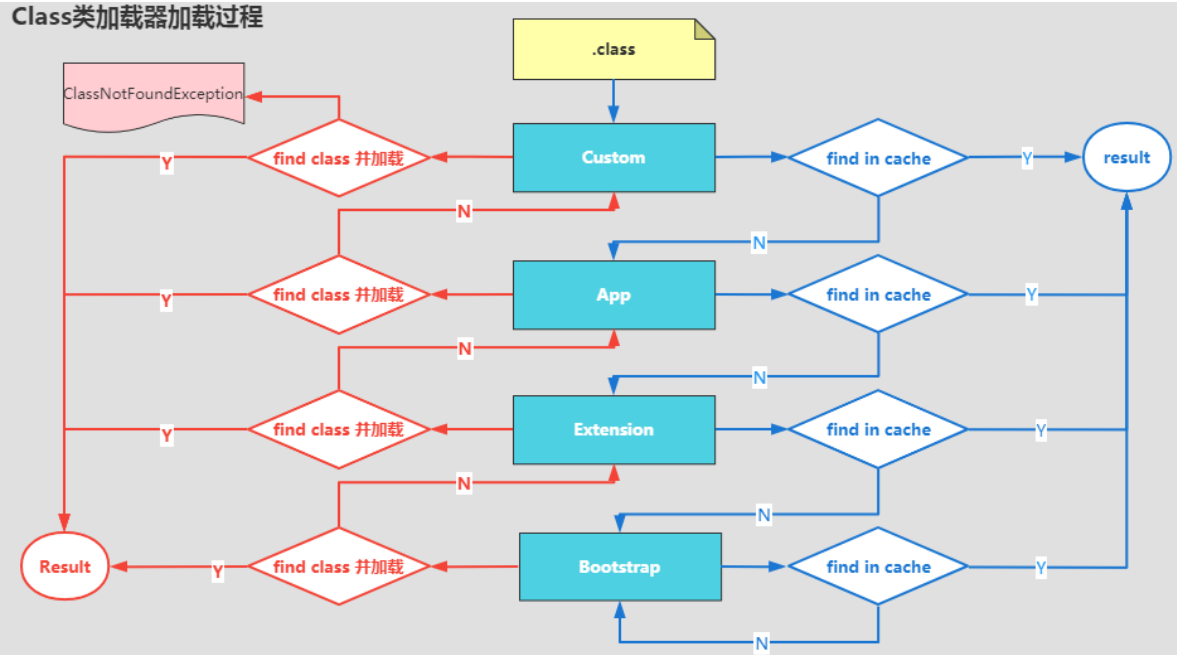
Custom 自定义的类加载器

Class 类加载器加载过程

双亲委派：从子到父的过程和从父到子的过程

过程：

class首先会从 Custom加载器（缓存 ——每个类加载器 维护一个list 用来管理已经加载的对象引用）中找，找到返回结果，找不到会从上层加载器APP中找，如果还是找不到再到Ext加载器中招，一直到Bootstrap加载器，(前面的find 都是从缓存中找)，如果缓存中没有，则find class 并加载，如果 bootstrap没有加载到，则交给下一层加载器 ext find class 并加载 ，直到custom 。如果custom 没有找到class,则报错 ClassNotFoundException异常



父加载器：没有直接关系，当前加载器加载不到内容的时候会找上层加载器

源码中 每一个加载器维护了一个 属性名师parent的ClassLoader，通过这个属性找对应关系

双亲委派注意的问题：

- 父加载器

  父加载器不是“类加载器的加载器”，也不是“类加载器的父类加载器”

- 双亲委派是一个孩子向父亲方向，然后父亲向孩子方向的双亲委派过程

面试：为啥要搞双亲委派

- 主要是为了安全

  假如没有双亲委派，我自己定义一个 java.lang.String ，这样自定classloader 则会加载到的是自己写的，不是oracle的，如果是双亲委派，则find cache 的时候 找到的jdk的String

- 其次是效率问题

如何打破双亲委派机？ 即不想用双亲委派机制

由于委培机制是在 ClassLoader 类中的 loadClass方法中完成的

通过parent.loadClass 进行向上查找 findClass 向下加载 完成

所以如果要想打破这个机制，则重写 loadClass 方法即可