深入理解ClassLo

2018年08月05日 23:28:46 zthgre

ClassLoader 顾名思义就是类加

- 负责将 Class 加载到 JVM 中
- 审查每个类由谁加载（父优先的
- 将 Class 字节码重新解析成 JV

类加载时机与过程

类从被加载到虚拟机内存中开始 ...包括了：加载、验证、准备、解析、初始化、使用和卸载这7个阶段。其中，和解析这三个部分统称为连接（

其中，加载、验证、准备、初始 ...加载过程必须按照这种顺序按部就班的“开始”（仅仅指的是开始，而非执行为这些阶段通常都是互相交叉的 ...中调用或者激活另一个阶段），而解析阶段则不一定（它在某些情况下可以在后再开始，这是为了支持Java语

什么情况下需要开始类加载过程 ...行约束，这点可以交给虚拟机的的具体实现自由把握，但是对于初始化阶段严格规定了如下几种情况，如果

1. 创建类的实例
2. 对类进行反射调用的时候，如
3. 当初始化一个类的时候，如果 ...其父类的初始化
4. 当虚拟机启动时，用户需要指 ...，虚拟机会先初始化这个主类
5. 当使用jdk1.7动态语言支持时 ...最后的解析结果REF_getstatic,REF_putstatic,REF_invokeStatic的方法句柄，并且这个方的类没有进行初始化，则需要

注意，对于这五种会触发类进行 ...很强烈的限定语：“有且只有”，这五种场景中的行为称为对一个类进行 主动引外，所有引用类的方式，都不会

特别需要指出的是，类的实例化

- 类的实例化是指创建一个类的实
- 类的初始化是指为类中各个 ...的过程，是类生命周期中的一个阶段。

下面是被动引用的几个例子（1）

```
1  /**
2   * jdk:1.8
3   * 通过子类引用父类的静
4   */
5  class  SuperClass{
6
7      static {
8          System.out.p
9      }
```

```
    public  static  int value=123;

}
class  SubClass extends  SuperClass{

    static {
        System.out.println("SubClass init!");
    }

}

public class Test {
    public static void main(String[] args)throws  Exception{
      System.out.println(SubClass.value);
      //输出:
      //SuperClass init!
      //123

    }
}
```

通过其子类来引用父类中定义的静态字段，只会触发父类初始化而不会触发子类的初始化。至于是否要触发子类的加载和验证，这个在虚拟机规范中并定，这点取决于虚拟机的具体实现。

(2)

```
/**
 * jdk：1.8
 * 通过数组定义来引用类，不会触发此类的初始化
 */
class  SuperClass{

    static {
        System.out.println("SuperClass init!");
    }
    public  static  int value=123;

}
class  SubClass extends  SuperClass{

    static {
        System.out.println("SubClass init!");
    }

}

public class Test {
    public static void main(String[] args)throws  Exception{
        SuperClass[] sca=new SuperClass[10];
        //无输出
    }
}
```

通过数组定义来引用类，不会触发此类的初始化。

(3)

```
/**
 * jdk:1.8
 * 常量在编译阶段会存入调用类的常量池，
 * 本质上并没有直接引用到定义常量的类，因此不会触发定义常量的类的初始化
 */
class ConstClass{
    static {
        System.out.println("ConstClass init!");
    }
    public static  final String HELLO = "hello";

}

```

```
14 public class Test {
15     public static void main(String[] args)throws  Exception{
16         System.out.println(ConstClass.HELLO);
17         //输出 hello
18     }
19 }
```

常量在编译阶段会存入调用类的常量池，本质上并没有直接引用到定义常量的类，因此不会触发定义常量的类的初始化

ClassLoader 类结构分析

```
1 public abstract class ClassLoader {
2
3   // The parent class loader for delegation
4   // Note: VM hardcoded the offset of this field, thus all new fields
5   // must be added *after* it.
6   private final ClassLoader parent;
7
8   //...
9 }
```

这里注意下有父加载器，这个我们再后面再来阐述，这里先有个印象。

以下是 ClassLoader 常用到的几个方法及其重载方法：

(1)

```
1 defineClass(String name, java.nio.ByteBuffer b,ProtectionDomain protectionDomain)
```

指定保护域（protectionDomain），把ByteBuffer的内容转换成 Java 类，这个方法被声明为final的。

(2)

```
1 defineClass(String name, byte[] b, int off, int len)
```

把字节数组 b中的内容转换成 Java 类，其开始偏移为off,这个方法被声明为final的。

(3)

```
1 //查找指定名称的类
2 findClass(String name)
```

(4)

```
1 //加载指定名称的类
2 loadClass(String name)
```

(5)

```
1 //链接指定的类
2 resolveClass(Class<?>)
```

其中 defineClass 方法用来将 字节流解析成 JVM 能够识别的 Class 对象，有了这个方法意味着我们不仅仅可以通过 class 文件实例化对象，还可以通过
例化对象，如果我们通过网络接收到一个类的字节码，拿到这个字节码流直接创建类的 Class 对象形式实例化对象。如果直接调用这个方法生成类的 C
这个类的 Class 对象还没有 resolve ，这个 resolve 将会在这个对象真正实例化时才进行。

defineClass 通常是和findClass 方法一起使用的，我们通过覆盖ClassLoader父类的findClass 方法来实现类的加载规则，从而取得要加载类的字节码，
defineClass方法生成类的Class 对象，如果你想在类被加载到JVM中时就被链接，那么可以接着调用另一个 resolveClass 方法，当然你也可以选择让J
么时候才链接这个类。

ClassLoader 的等级加载机制

Java默认提供的三个ClassLoader

BootStrap ClassLoader

称为启动类加载器，是Java类加载层次中最顶层的类加载器，负责加载JDK中的核心类库，如：rt.jar、resources.jar、charsets.jar等，这个ClassLoad自己控制的，需要加载哪个类，怎么加载都是由JVM自己控制，别人也访问不到这个类，所以BootStrap ClassLoader不遵循委托机制(后面再阐述什么制)，没有子加载器。

下面是测试 BootStrap ClassLoader 加载的哪些文件：

```
/**
 * BootStrap ClassLoader 加载的文件
 */
public class Test {
    public static void main(String[] args)throws  Exception{
        System.out.println(System.getProperty("sun.boot.class.path"));
    }
}
```

输出：

```
C:\Program Files\Java\jre1.8.0_151\lib\resources.jar;
C:\Program Files\Java\jre1.8.0_151\lib\rt.jar;
C:\Program Files\Java\jre1.8.0_151\lib\sunrsasign.jar;
C:\Program Files\Java\jre1.8.0_151\lib\jsse.jar;
C:\Program Files\Java\jre1.8.0_151\lib\jce.jar;
C:\Program Files\Java\jre1.8.0_151\lib\charsets.jar;
C:\Program Files\Java\jre1.8.0_151\lib\jfr.jar;
C:\Program Files\Java\jre1.8.0_151\classes
```

EtxClassLoader

称为扩展类加载器，负责加载Java的扩展类库，Java 虚拟机的实现会提供一个扩展库目录，该类加载器在此目录里面查找并加载 Java 类。默认加载JAVA_HOME/jre/lib/ext/目下的所有jar。

```
/**
 * EtxClassLoader 加载文件
 */
public class Test {
    public static void main(String[] args)throws  Exception{
        System.out.println(System.getProperty("java.ext.dirs"));
    }
}
```

输出：

```
C:\Program Files\Java\jre1.8.0_91\lib\ext;
```

AppClassLoader

称为系统类加载器，负责加载应用程序classpath目录下的所有jar和class文件。一般来说，Java 应用的类都是由它来完成加载的。可以通过ClassLoader.getSystemClassLoader()来获取它。我们可以通过System.getProperty(“java.class.path”) 来查看 classpath。

除了引导类加载器（BootStrap ClassLoader）之外，所有的类加载器都有一个父类加载器，对于系统提供的类加载器来说，系统类加载器(如：AppCla父类加载器是扩展类加载器（EtxClassLoader），而扩展类加载器的父类加载器是引导类加载器；对于开发人员编写的类加载器来说，其父类加载器是载器 Java 类的类加载器。因为类加载器 Java 类如同其它的 Java 类一样，也是要由类加载器来加载的。一般来说，开发人员编写的类加载器的父类加类加载器。类加载器通过这种方式组织起来，形成树状结构。树的根节点就是引导类加载器。

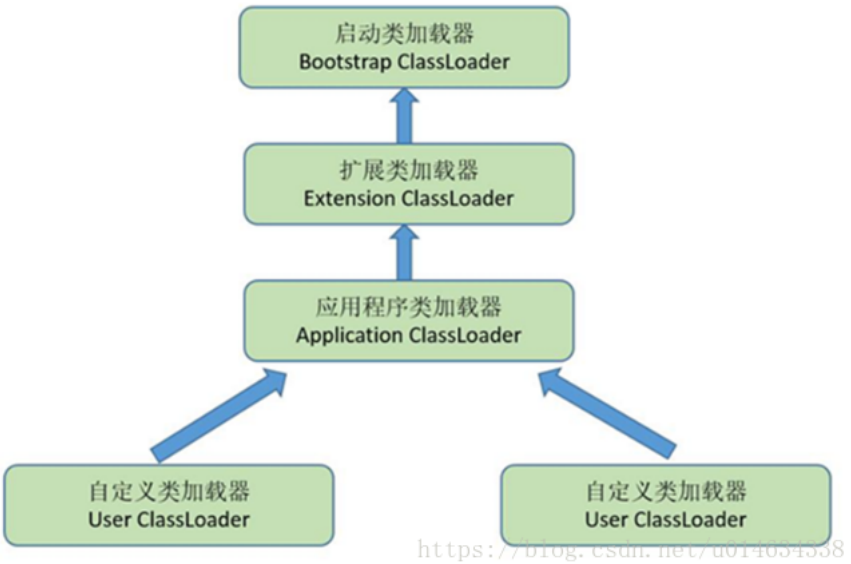
ClassLoader加载类的原理

ClassLoader使用的是双亲委托模型来搜索加载类的

双亲委托模型

ClassLoader使用的是双亲委托机制来搜索加载类的，每个ClassLoader实例都有一个父类加载器的引用（不是继承的关系，是一个组合的关系），虚加载器（Bootstrap ClassLoader）本身没有父类加载器，但可以用作其它ClassLoader实例的的父类加载器。当一个ClassLoader实例需要加载某个类时亲自搜索某个类之前，先把这个任务委托给它的父类加载器，这个过程是由上至下依次检查的，首先由最顶层的类加载器Bootstrap ClassLoader试图加加载到，则把任务转交给Extension ClassLoader试图加载，如果也没加载到，则转交给App ClassLoader 进行加载，如果它也没有加载得到的话，则发起者，由它到指定的文件系统或网络等URL中加载该类。如果它们都没有加载到这个类时，则抛出ClassNotFoundException异常。否则将这个找到的类的定义，并将它加载到内存当中，最后返回这个类在内存中的Class实例对象。

类加载器双亲委托模型：



双亲委托模型好处

因为这样可以避免重复加载，当父亲已经加载了该类的时候，就没有必要 ClassLoader再加载一次。考虑到安全因素，我们试想一下，如果不使用这种
我们就可以随时使用自定义的String来动态替代java核心api中定义的类型，这样会存在非常大的安全隐患，而双亲委托的方式，就可以避免这种情况，
经在启动时就被引导类加载器（Bootstrcp ClassLoader）加载，所以用户自定义的ClassLoader永远也无法加载一个自己写的String，除非你改变JDK中
搜索类的默认算法。

类与类加载器

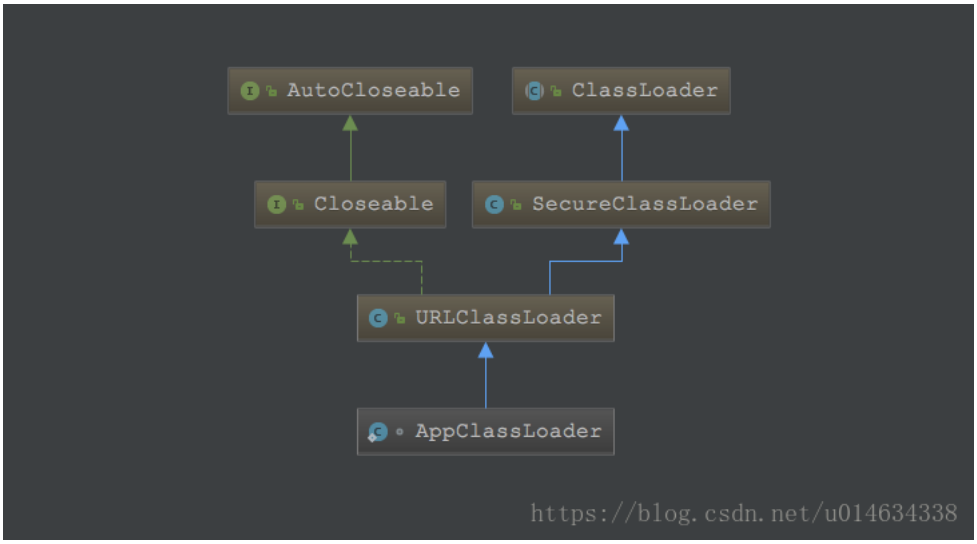
类加载器虽然只用于实现类的加载动作，但它在Java程序中起到的作用却远远不限于类加载阶段。对于任意一个类，都需要由加载它的类加载器和这个
确立其在Java虚拟机中的唯一性，每一个类加载器，都拥有一个独立的类名称空间。这句话可以表达更通俗一些：比较两个类是否"相等"，只有再这两
个类加载器加载的前提下才有意义，否则，即使这两个类来源于同一个**Class** 文件，被同一个虚拟机加载，只要加载它们的类加载器不同，那这两个
相等。

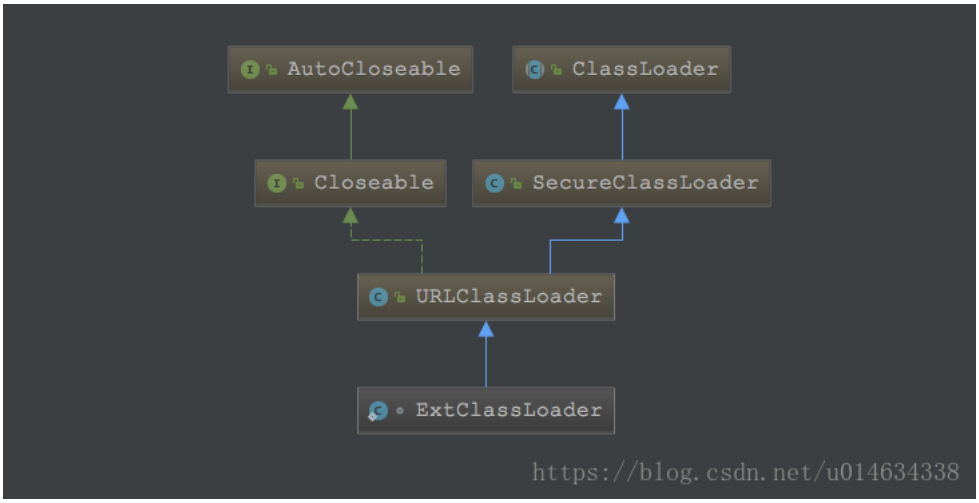
类加载器源码分析

Launcher

```
1  public Launcher() {
2      ExtClassLoader localExtClassLoader;
3      try {
4          // 扩展类加载器
5          localExtClassLoader = ExtClassLoader.getExtClassLoader();
6      } catch (IOException localIOException1) {
7          throw new InternalError("Could not create extension class loader", localIOException1);
8      }
9      try {
10         // 应用类加载器
11         this.loader = AppClassLoader.getAppClassLoader(localExtClassLoader);
12     } catch (IOException localIOException2) {
13         throw new InternalError("Could not create application class loader", localIOException2);
14     }
15     // 设置AppClassLoader为线程上下文类加载器
16     Thread.currentThread().setContextClassLoader(this.loader);
17     // ...
18
19     static class ExtClassLoader extends java.net.URLClassLoader
20     static class AppClassLoader extends java.net.URLClassLoader
21 }
```

Launcher初始化了ExtClassLoader和AppClassLoader，并将**AppClassLoader**设置为线程上下文类加载器。
ExtClassLoader和AppClassLoader都继承自URLClassLoader，而最终的父类则为ClassLoader,看看它们的类层次：





初始化AppClassLoader时传入了ExtClassLoader实例，当我们进入源码跟踪，会来到URLClassLoader 中

```
1  public URLClassLoader(URL[] urls, ClassLoader parent,
2                        URLStreamHandlerFactory factory) {
3      super(parent);
4      // this is to make the stack depth consistent with 1.1
5      SecurityManager security = System.getSecurityManager();
6      if (security != null) {
7          security.checkCreateClassLoader();
8      }
9      ucp = new URLClassPath(urls, factory);
10     acc = AccessController.getContext();
11 }
```

可以得知，这个**ExtClassLoader 作为了AppClassLoader 的parent**，在前面ClassLoader 源码中，我们知道有个parent 字段，这里就是在初始化这个

双亲委派

要理解双亲委派，可以查看ClassLoader.loadClass方法:

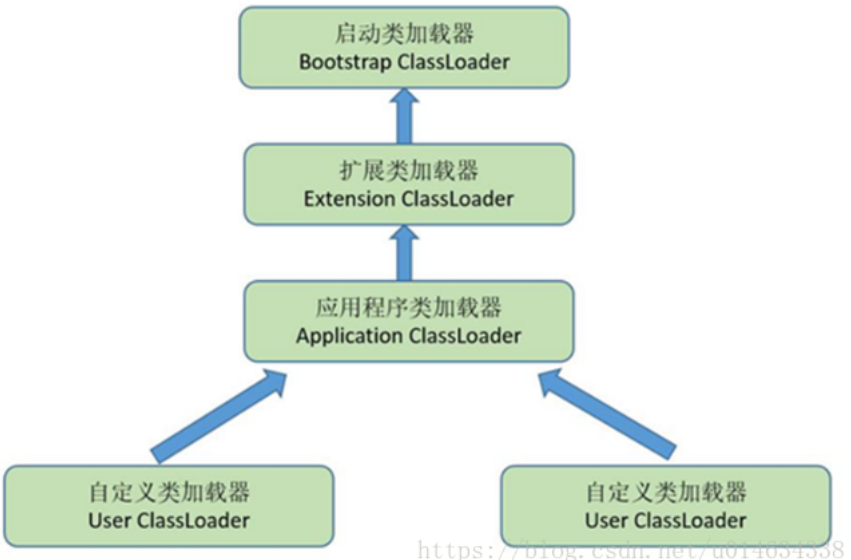
```
1  protected Class<?> loadClass(String name, boolean resolve) throws ClassNotFoundException {
2      synchronized (getClassLoadingLock(name)) {
3          // 检查是否已经加载过
4          Class<?> c = findLoadedClass(name);
5          if (c == null) { // 没有被加载过
6              long t0 = System.nanoTime();
7              // 先委派给父类加载器加载
8              try {
9                  if (parent != null) {
```

```
                    c = parent.loadClass(name, false);
                } else {
                    //如果父加载器不存在，则委托给启动类加载器 加载
                    c = findBootstrapClassOrNull(name);
                }
            } catch (ClassNotFoundException e) {
                // ClassNotFoundException thrown if class not found
                // from the non-null parent class loader
            }

            if (c == null) {
                // 如果父类加载器无法加载，自身才尝试加载
                long t1 = System.nanoTime();
                c = findClass(name);

                // this is the defining class loader; record the stats
                sun.misc.PerfCounter.getParentDelegationTime().addTime(t1 - t0);
                sun.misc.PerfCounter.getFindClassTime().addElapsedTimeFrom(t1);
                sun.misc.PerfCounter.getFindClasses().increment();
            }
        }
        if (resolve) {
            resolveClass(c);
        }
        return c;
    }
}
```

先检查是否已经被加载过，若没有加载则调用父加载器的loadClass() 方法，若父加载器为空，则默认使用启动类加载器作为父加载器。如果父加载器自己的findClass 方法进行加载,因此到这里再次证明了类加载器的过程：



总结

关于类加载器网上有很多的文章，记录下来，也算是一个自己总结的过程，有时候看很多遍，不如实实在在的写一遍。

参考

深入理解Java 虚拟机
深入分析Java Web技术内幕
深入理解JVM之ClassLoader
详细深入分析 Java ClassLoader 工作机制

想对作者说点什么

一看你就懂，超详细java中的ClassLoader详解

13.2万

本篇文章已授权微信公众号 guolin_blog （郭霖）独家发布 ClassLoader翻译过来就是类加载器，普通的java开发者… 来自： frank 的专栏

classLoader机制

253

一、什么是ClassLoader? 大家都知道，当我们写好一个Java程序之后，不是管是CS还是BS应用，都是由若… 来自： weixin_40813139的博客

ClassLoader工作机制

489

基本信息 作者：kaedea 项目：android-dynamical-loading 类加载器ClassLoader 早期使用过Eclipse等Java编写的… 来自： JackChan

ClassLoader机制

1137

ClassLoader是用来加载class文件到JVM，以供程序使用。我们知道，java程序可以动态加载类定义，而这个动态… 来自： zh23862691的专栏

【JAVA笔记——术】Java ClassLoader类加载机制详解

2166

Java ClassLoader三种类加载器WIki : Java_ClassloaderJAVA类加载器实现了一部分 JRE加载JAVA CLASSES到 J… 来自： 从start到running

深入分析ClassLoader工作机制

3802

ClassLoader 较为深入分析。from 加载CLASS到JVM中，审查每个类应该由谁加载，父优先的等级加载机制。加载… 来自： 思念

浅析ClassLoader工作机制

42

浅析ClassLoader工作机制ClassLoader顾名思义就是类加载器，负责将Class字节码文件加载到JVM中。如何加载cl… 来自： antman_spring的博客

java 的 ClassLoader 类加载机制详解

3135

一个程序要运行，需要经过一个编译执行的过程： Java的编译程序就是将Java源程序 .java 文件 编译为JVM可执行… 来自： 请叫我王老魔

下载 **ClassLoader机制详解** 05-24

由osgi引出的classLoader的大总结（整理理解ClassLoader）

深度分析Java的ClassLoader机制（源码级别）

2195

http://www.hollischuang.com/archives/199 写在前面：Java中的所有类，必须被装载到jvm中才能运行，这个装载工… 来自： zdy0_2004的专栏

Python 凭什么碾压Java、C++等老牌语言?

4.1万

随着计算机语言的发展，Python也跻身于语言排行的常青树。要是说Python是最目前最火爆的语言，应该没有人反… 来自： CSDN学院

Java/JDK 8 新特性1.8对于1.7做了哪些优化/改进

8074

ava 8 新特性 Java 8 (又称为 jdk 1.8) 是 Java 语言开发的一个主要版本。 Oracle 公司于 2014 年 3 月 18 日发布 Ja… 来自： hotpots--火锅技术栈

Java 类加载器（ClassLoader）的实际使用场景

540

资源隔离 热部署 代码保护 Tomcat容器，每个WebApp有自己的ClassLoader,加载每个WebApp的ClassPath路径上… 来自： 落叶翩翩的CSDN博客

classloader机制研究(2) -- 应用场景 802
本篇文章实际是的摘译，具体来讲是摘译了其中的第三部分。这个章节的题目翻译成中文的话应该是: class loaders… 来自： shuimuniao的专栏

JDK1.8 动态代理机制及源码解析 2073
1.静态代理 核心思想：代理对象通过持有需要被代理类的实例，实现代理方式。 参考链接： http://www.cnblogs.co… 来自： 灵小帝的博客

Java类加载机制与Tomcat类加载器架构 7669
Java类加载机制 类加载器 虚拟机设计团队把类加载阶段中的“通过一个类的全限定名来获取描述此类的二进制字节… 来自： Hopefully Sky的博客

mac常用软件工具 140
homeBrew: https://www.wikiwand.com/zh-hans/Homebrew#/%E6%9C%BA%E5%88%B6 来自： WitsMakeMen的专栏

在本地将spark作业运行到远程集群 6106
在本地IDE里直接运行spark程序操作远程集群一般运行spark作业的方式有两种： 本机调试，通过设置master为loc… 来自： 乔的博客

一道面试题搞懂JVM类加载机制 135
JVM(四)—一道面试题搞懂JVM类加载机制 有这样一道面试题：class Singleton{ private stat... 来自： TuxedoLinux的博客

C++重载机制 215
原文地址：http://huqunxing.site/2016/09/08/C++%E9%87%8D%E8%BD%BD%E6%9C%BA%E5%88%B6/C++ 允… 来自： 徐小小小hi的博客

深入理解Redis的持久化机制和原理 800
Redis是一种面向“key-value”类型数据的分布式NoSQL数据库系统，具有高性能、持久存储、适应高并发应用场景… 来自： 逍遥飞鹤的专栏

详解Java Socket的工作机制 59
Socket的来龙去脉 下面的分析主要是参阅了计算机网络（谢希仁第7版）进行总结的，从系统调用—>应用… 来自： pjmike的博客

下载 **理解Java ClassLoader机制** 10-27
理解Java ClassLoader机制

【深入理解Java虚拟机】类加载机制 8274
本文内容来源于《深入理解Java虚拟机》一书，非常推荐大家去看一下这本书。本系列其他文章：【深入理解Java… 来自： 开心阳

深刻理解HDFS工作机制 1779
深入理解一个技术的工作机制是灵活运用和快速解决问题的根本方法，也是唯一途径。对于HDFS来说除了要明白… 来自： csao204282的博客

深度分析 Java 的 ClassLoader 机制（源码级别） 852
Java中的所有类，必须被装载到jvm中才能运行，这个装载工作是由jvm中的类装载器完成的，类装载器所做的工作… 来自： Jason的博客

理解Java ClassLoader机制 |用Java说话，人气战胜时间！Come On 197
理解Java ClassLoader机制 |用Java说话，人气战胜时间！Come On 我在参加一个比赛。 欢迎大家都来我的网站参… 来自： likaiwalkman@csdn - Co…

Java中的ClassLoader 动态加载机制 7577
前言： Android中的动态加载机制能更好的优化我们的应用，同时实现动态的更新，这就便于我们管理我们的应用… 来自： lxj_time的博客

classloader的加载流程与特性 142
java程序不是一个可执行文件而是由许多独立的Class（类）文件组成的，每一个Class对应一个类文件，这些类文… 来自： A_Story_of_Yonosuke的…

Java 代理机制 155

本文纯属搬运，参考以下文章（侵删，请联系）： Java代理机制与hook: http://www.zengye.cc/2016/05/01/java%E… 来自： MachineRandy

【6】-BAT面试之操作系统内存详解 18

本文主要参考两篇博客，读后整理出来，以供大家阅读，链接如下： http://blog.jobbole.com/95499/?hmsr=toutiao.i… 来自： 世上只有一种英雄主义

下载 **深入理解Spark:核心思想与源码分析（完整版）** 09-26

在深入了解一个系统的原理、实现细节之前，应当先准备好它的源码编译环境、运行环境。如果能在实际环境安装和运行Spark，显然能够提升读者对于Spark的一些感受，对系统…

深入理解JVM的内存结构及GC机制 7920

一、前言 JAVA GC（Garbage Collection，垃圾回收）机制是区别C++的一个重要特征，C++需要开发者自己实… 来自： EnjoyAndroid的博客

深入理解驱动开发中的机制与策略 145

机制mechanism，策略policy。如果你看过《linux device drivers》，里面给出了大概的介绍。机制提供了干什么(do… 来自： 私房菜之 --学--无--止--境--

深入理解系列之JDK8下JVM虚拟机（1）——JVM内存组成 2826

今天开始谈论一些JVM虚拟机的知识。其实在前面叙述中多多少少已经附带提起了JVM相关的知识，如类加载、多… 来自： 站在人文与技术的交汇点

详细深入分析 Java ClassLoader 工作机制 2578

详细深入分析 Java ClassLoader 工作机制 什么是 ClassLoader ClassLoader 作用 1ClassLoader 类结构分析 2Clas… 来自： http://www.54tianzhishen…

深入分析Java ClassLoader原理 12.7万

一、什么是ClassLoader? 大家都知道，当我们写好一个Java程序之后，不是管是CS还是BS应用，都是由若… 来自： 技术改变生活

ClassLoader 机制 1112

我们知道，Java利用ClassLoader将类载入内存，并且在同一应用中，可以有很多个ClassLoader，通过委派机制，… 来自： fdxganli的专栏

JVM的Classloader机制 377

原文链接：https://segmentfault.com/q/1010000000155690/a-1020000000155732 首先，你需要了解一下JVM的Cl… 来自： ice-wee的专栏

java 深入分析ClassLoader工作机制 1650

ClassLoader顾名思义就是类加载器，负责将Class加载到JVM中，它就好比开会时 门口的接待员，负责给进入会场… 来自： Code-lover's Learning No…

java ClassLoader工作机制 4051

java应用环境中不同的class分别由不同的ClassLoader负责加载。 一个jvm中默认的classloader有Bootstrap ClassL… 来自： Delion

下载 **【图解版】深入分析ClassLoader类加载工作机制** 08-16

【图解版】深入分析ClassLoader类加载工作机制，从原理到JVM的装载过程，详情分析了ClassLoader加载类以及自定义类加载器的过程，不可用于商业用途，如有版权问题，请…

JavaWeb技术内幕六：深入分析classloader工作机制 424

classloader顾名思义就是类加载器，负责将class加载到jvm中。 事实上，classloader除了能将class加载到jvm中，… 来自： 程序员江湖

Android中ClassLoader和java中ClassLoader有什么关系和不同 206

Android，ClassLoader，java，关系，不同 来自： bolang789的博客

线程池原理（JDK1.8） 73

Java中的线程池 ThreadPoolExecutor是线程池类。对于线程池，可以通俗的将它理解为”存放一定数量线程的一个… 来自： 食鱼酱的博客

深入理解bootloader_3----- ARM体系结构 910

深入理解bootloader_3—— ARM体系结构 来自： 静水鱼游的博客

深入理解Java类加载器(ClassLoader) 9.1万

【版权申明】未经博主同意，谢绝转载！（请尊重原创，博主保留追究权） http://blog.csdn.net/javazejian/article/d… 来自： zejian的博客

ClassLoader的分析与使用 1461

原文地址 md直接复制 乱码请查看原文～～ 深入学习ClassLoader原理与学习自定义ClassLoader的使用 JAVA自带… 来自： Lewis的博客

Java虚拟机--ClassLoader（十九） 627

目录：ClassLoader工作在Class装载的加载阶段，主要作用是从系统外部获得Class二进制数据流 知识点的梳理… 来自： 老赫的私人厨房

深入浅出ClassLoader, 你真的了解ClassLoader吗? 7528

http://ifeve.com/classloader/ Dedicate to Molly. 你真的了解ClassLoader吗? 这篇文章翻译自zeroturnar... 来自： liangxw1的专栏

webstorm 2018 激活破解方法大全 677881

webstorm 作为最近最火的前端开发工具,也确实对得起那个价格,但是秉着勤俭节约的传统美德,我们肯定是能省则省… 来自： 唐大帅的编程之路

【C#从入门到遛弯】第十二章 · 虚方法和抽象类 927

1.抽象成员必须标记为abstract,并且不能有任何实现。 2.抽象成员必须在抽象类中。 3.抽象类不能被实例化 4.子… 来自： 唐三十胖子的博客

史上最简单的 SpringCloud 教程 | 终章 1238117

转载请标明出处： http://blog.csdn.net/forezp/article/details/70148833 本文出自方志朋的博客 错过了这一篇，你可… 来自： 方志朋的专栏

字符输入的区别及不同的排序算法 865

目前已知的三种输入函数 scanf（）特点输入种类繁多，要求按照规定格式输入，期间不能加入空格（会数据丢失）… 来自： 唐三十胖子的博客

批处理添加字段和删除字段 16694

在平时的工作中你或许会遇到这样的问题，给某个文件夹下面的所有的图片添加同一个字符串，如果一个一个去F2… 来自： pyf_914406232的博客

整理了10个干净、好用的BT、磁力链搜索网站给大家 95440

现在越来越流行在线看视频了，但是对于我得收藏癖爱好者，还是希望可以有比较好的资源网站的，尤其是种子、… 来自： YXAPP的技术分享

排序算法（六）快速排序验证性实验 10560

请创建一个一维整型数组用来存储待排序关键码，关键码从数组下标为1的位置开始存储，下标为0的位置不存储关… 来自： 青衣煮茶

户外羽绒服排行榜,它才是女人最好的嫁妆! 4902

Public Function Encrypt(src As String) As String -----加密 Dim i As Integer Dim aStr As String Dim ... 来自： john_dung的博客

容器将成为下一个“Linux” 13044

... 来自： Docker的专栏

微服务Springcloud超详细教程+实战（六） 3739

如在文档中遇到什么问题请联系作者 QQ：1172796094 本人正在找深圳Java实习工作，求大佬带飞 ————————…

解决汉字显示问题的一种方法: 5185

解决汉字显示问题的一种方法： 在本地文件中添加一个txt文件，文件的格式是前面写序号后面写汉字的格式，然… 来自： pyf_914406232的博客

2018最新Web前端经典面试试题及答案 355706

本篇收录了一些面试中经常会遇到的经典面试题以及自己面试过程中遇到的一些问题，并且都给出了我在网上收集… 来自： wdlhao的博客

很黄很暴力的十个网站 61989

13岁的北京学生张某，在去年12月27日19时新闻联播一则关于净化网络视听的新闻里，接受采访时说的话激起了轩… 来自： Kinb_huangwei的专栏

Postman 使用方法详解 186185

一、Postman背景介绍 用户在开发或者调试网络程序或者是网页B/S模式的程序的时候是需要一些方法来跟踪网页… 来自： fxbin123的博客

微服务Springcloud超详细教程+实战（五）
4691
如在文档中遇到什么问题请联系作者 QQ：1172796094 本人正在找深圳Java实习工作，求大佬带飞 ————————…

微服务Springcloud超详细教程+实战（一）
4937
如在文档中遇到什么问题请联系作者 QQ：1172796094 本人正在找深圳Java实习工作，求大佬带飞 ————————…

锤子变天？| 畅言
3538
作者 | 小谦 责编 | 郭芮 或许是因为生不逢时，锤子科技这次要彻底变天了。 12月12日，在这个非常重要的购物促… 来自： CSDN资讯

微服务Springcloud超详细教程+实战（十）
2480
本人正在找深圳Java实习工作，求大佬带飞 QQ：1172796094 如在文档中遇到什么问题请联系作者 ————————…

pyCharm最新2018激活码
1244940
本教程对jetbrains全系列可用例：IDEA、WebStorm、phpstorm、clion等 因公司的需求，需要做一个爬取最近上映… 来自： 昌昌

（二）MyBatis核心组件（配图详解＆代码实现）
13185
MyBatis的核心组件分为4个部分 SqlSessionFactoryBuilder（构造器）：根据xml或java代码生成SqlSessionFactory… 来自： 青衣煮茶

【《Unity Shader入门精要》 提炼总结】(十)第十章·法线贴图概念&切线空间下法线Shader实现&模型空…
971
本文由@唐三十胖子出品，转载请注明出处。 文章链接：https://blog.csdn.net/iceSony/article/details/8459187… 来自： 唐三十胖子的博客

微服务Springcloud超详细教程+实战（八）
3642
如在文档中遇到什么问题请联系作者 QQ：1172796094 本人正在找深圳Java实习工作，求大佬带飞 ————————…

docker入门+结合微服务实战（五）
4367
如在文档中遇到什么问题请联系作者 QQ：1172796094 本人正在找深圳Java实习工作，求大佬带飞 ————————…

基于CAS线程安全实现计数器
7901
在多线程环境下，实现一个CAS原理的线程安全的技术器，并与不使用CAS算法的计数器进行比较。 package com… 来自： 青衣煮茶

门罗币 xmr 超级详细的CPU xmr挖矿教程
73073
门罗币 xmr 最详细的CPU 挖矿教程 基础 CUP 挖矿教程 如何挖矿？ Step1:获得一个钱包地址 钱包分为两个部分讲,… 来自： qq_39863517的博客

SpringBoot整合ActiveMQ消息队列
1139
首先要讲什么是ActiveMQ：AciveMQ是Apache出品的目前最流行，能力强劲的开源消息总线 主要功能： 1、 解决… 来自： 熊局长的博客

微服务Springcloud超详细教程+实战（二）
5294
远程调用方式 无论是微服务还是分布式服务（都是SOA，都是面向服务编程），都面临着服务间的远程调用。那么…

Jenkins使用问题记录
3035
1. 启动 使用Jenkins的版本为2.138.3，下载war包后启动即可运行： # 指定使用8080端口，可自定义 java -jar jenki… 来自： jacksonary的博客

教你用TensorFlow实现手写数字识别
1847
弱者用泪水安慰自己，强者用汗水磨练自己。 这段时间因为项目中有一块需要用到图像识别，最近就一直在炼丹，… 来自： 流月的博客

docker入门+结合微服务实战（八）
3912
如在文档中遇到什么问题请联系作者 QQ：1172796094 本人正在找深圳Java实习工作，求大佬带飞 ————————…

微服务Springcloud超详细教程+实战（九）
3220
如在文档中遇到什么问题请联系作者 QQ：1172796094 本人正在找深圳Java实习工作，求大佬带飞 ————————…

docker入门+结合微服务实战(一)
5008
docker入门（一） 如在文档中遇到什么问题请联系作者 QQ：1172796094 本人正在找深圳实习工作，求大佬带飞 …

java练习总结
2155
记录平时使用java时发现的一些细节问题。 来自： qq_33699659的博客

史上最全Java面试题（带全部答案）
118268
今天要谈的主题是关于求职，求职是在每个技术人员的生涯中都要经历多次。对于我们大部分人而言，在进入自己… 来自： 林老师带你学编程

java缓冲区 4884

1 缓冲区的分类 ByteBuffer CharBuffer ShortBuffer IntBuffer LongBuffer FloatBuffer DoubleBuffer 2 ByteBuffe... 来自： weixin_43694144的博客

（一）使用IDEA创建Maven项目和Maven使用入门（配图详解） 6264

本文详解的讲解了使用IDEA创建Maven项目，及Maven的基础入门。 1、打开IDEA，右上角选择File->Ne… 来自： 青衣煮茶

【Android学习】第一章 · 安卓项目目录结构 1326

src: java源码所在目录 gen:自动生成的资源id（不能修改） Android.jar：导入安卓jar包才能使用安卓的api Android … 来自： 唐三十胖子的博客

军事理论课答案（西安交大版） 1072234

1.1 1 【单选题】我国陆地领土面积排名世界第几？（C） A、1 B、2 C、3 D、4 2 【单选题】以下哪个国家不属于… 来自： ling_wang的博客

（二）Maven的坐标和依赖＆利用Maven实现邮件发送 13046

本文中将《Maven实战》中对坐标和依赖的定义展示给初学Maven的程序猿们，并加上书中实例展示，具体详细请… 来自： 青衣煮茶

MySql数据库优化必须注意的四个细节（方法） 14115

MySQL 数据库性能的优化是 MySQL 数据库发展的必经之路， MySQL 数据库性能的优化也是 MySQL 数据库前… 来自： 青衣煮茶