Java Class Loader

目录

**[1 ClassLoader简介 1](#_Toc14734)**

**[2 ClassLoader种类 1](#_Toc17382)**

**[3 Class装载过程 2](#_Toc674)**

[3.1 加载(Loading) 2](#_Toc4948)

[3.2 连接(Linking) 3](#_Toc2047)

[3.3 初始化(Initialization) 3](#_Toc19817)

**[4 Class的加载方式 3](#_Toc3038)**

[4.1 隐式加载 3](#_Toc31124)

[4.2 显式加载 3](#_Toc30114)

[4.3 loadClass和forName区别 3](#_Toc12439)

**[5 自定义ClassLoader热更新 3](#_Toc13896)**

[5.1 自定义ClassLoader 3](#_Toc30136)

[5.2 Api方式热更新调用 5](#_Toc10577)

[5.3 两个Class在什么情况下是相同的 5](#_Toc20089)

[5.4 破坏双亲委派自定义ClassLoader 6](#_Toc7704)

[5.5 面向抽象类/接口 Api方式热更新调用 6](#_Toc13605)

**[6 游戏自定义ClassLoader问题 7](#_Toc21678)**

[6.1 全部错误信息 7](#_Toc9396)

[6.2 游戏内热更新ClassLoader结构图 7](#_Toc21166)

[6.3 游戏内热更新大概流程 7](#_Toc12495)

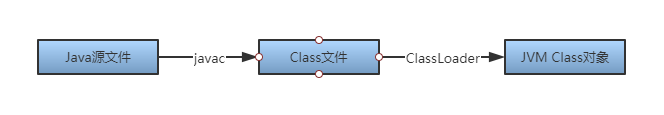
[6.4 出异常流程 8](#_Toc22445)

[6.5 如何必定出现该错误 9](#_Toc29440)

**[7 参考资料 11](#_Toc13873)**

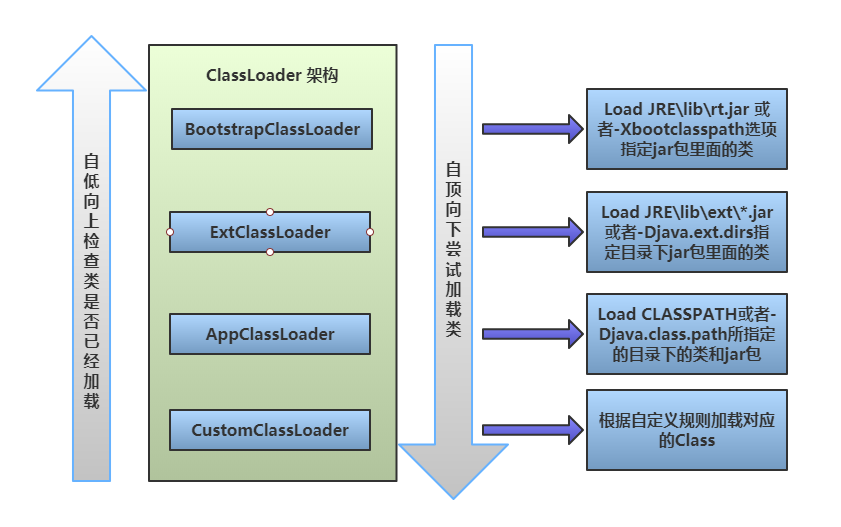
# ClassLoader简介

* 将Class文件加载到内存
* 并对数据进行校验、转换解析、初始化
* 最终构建成可以被虚拟机直接使用的Class对象
* 如下图



# ClassLoader种类

* BootStarpClassLoader: C++编写，加载核心库java.\*，有以下两个来源
  + JRE\lib\rt.jar
  + -Xbootclasspath 选项指定jar包里面的类
* ExtClassLoader: java编写，加载扩展库javax.\*，有以下两个来源
  + JRE\lib\ext\\*.jar
  + -Djava.ext.dirs
* AppClassLoader: java编写，加载程序所在目录
  + classpath路径里面的class，优先-Djava.class.path
  + -Djava.class.path，该配置可使用系统环境变量配置
* 自定义ClassLoader: java编写，定制化加载
* 架构如下图



* 该种加载方式称为ClassLoader双亲委派
  + 避免同一个类重复加载
  + 安全性问题

# Class装载过程

## 加载(Loading)

* 根据类全路径名加载二进制流
  + 从文件
  + 网络
  + 字节码生成。。。。
* 解析二进制流为运行时数据结构
* 构建Class对象

## 连接(Linking)

* 验证(Verification)，验证二进制流是否符合 class字节码规范
  + 魔数是否对的上
  + 字节码版本是否可以运行在当前jvm运行
  + 基本的语法规则验证
  + 基本的指令验证
  + 。。。。更详细的可以阅读 深入理解java虚拟机JVM高级特性与最近实践
* 准备(Preparation)
  + 为静态变量分配内存，并且进行初始化
* 解析(Resolution)
  + 将常量池的符号引用转换为直接引用
  + 该阶段只有在使用到才会发生，包括访问权限校验

## 初始化(Initialization)

* 执行clinit()函数
* 对静态变量进行初始化为程序设定的值
* 执行静态代码块

# Class的加载方式

## 隐式加载

* new

## 显式加载

* loadClass，forName

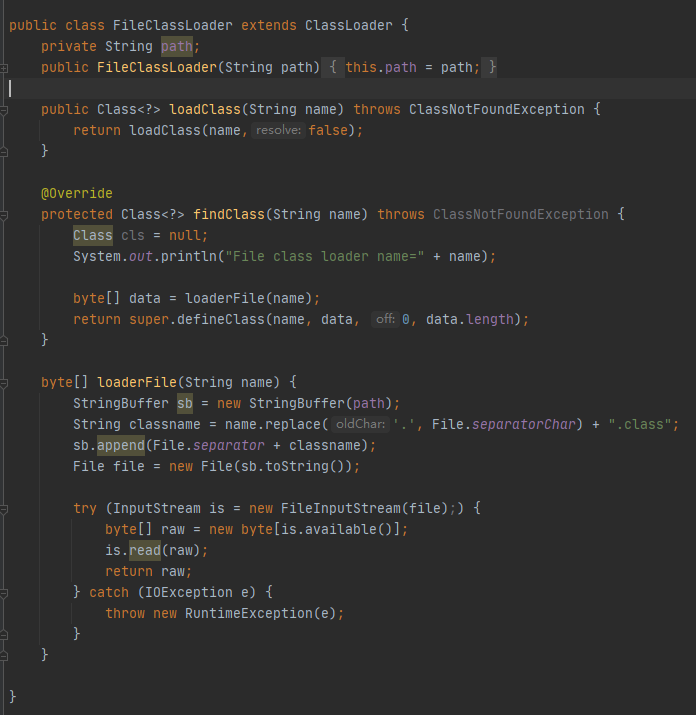
## loadClass和forName区别

* loadClass不会执行 初始化 过程，也就是执行静态代码块、变量赋值为程序设定
* forName 会执行 初始化 过程

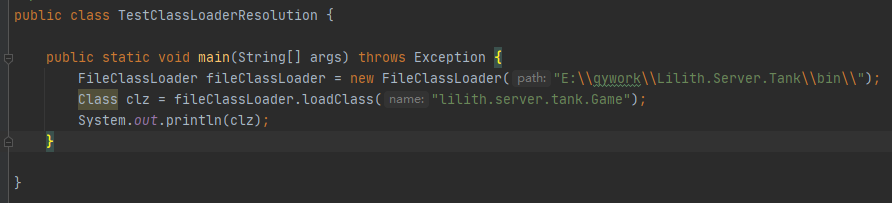
# 自定义ClassLoader热更新

## 自定义ClassLoader

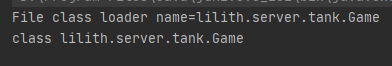
* 新建FileClassLoader继承ClassLoader，重写findClass，如下图



* 测试代码，如下图



* 输出

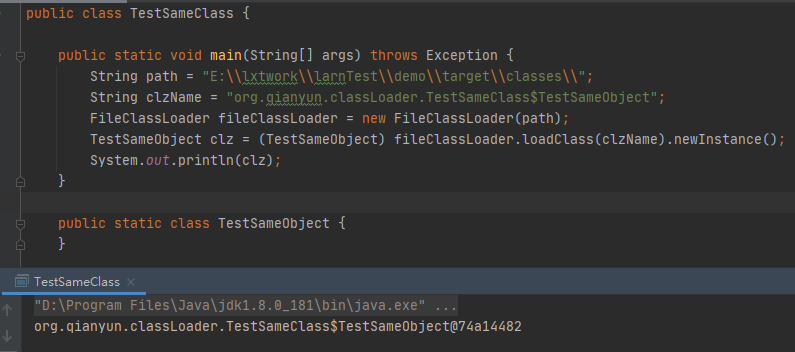


## Api方式热更新调用

* 使用加载出来的class，只能用反射的方式调用，这种方式非常不便利
* 我们需要转换成 A a = (A)loaderClass(“B”).newInstance()
* 这种写法才能愉快的写代码
* 强转就涉及到一个问题，两个Class在什么情况下是相同的，只有相同的Class才能进行强转

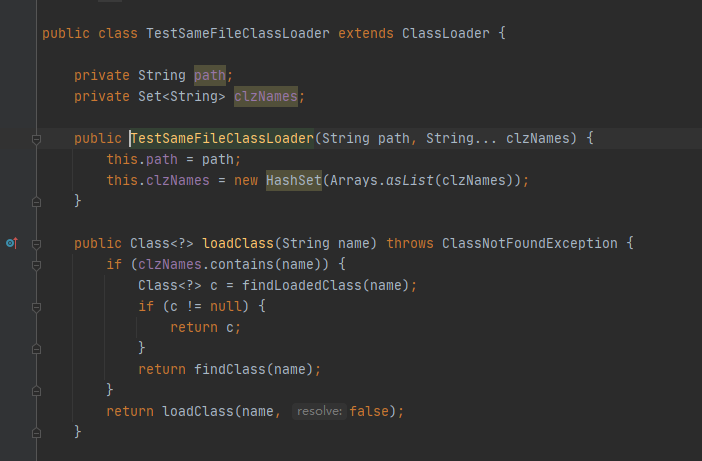
## 两个Class在什么情况下是相同的

* 两个Class只有满足以下情况才是相同的
  + 全路径名一样
  + 采用同一个实例的ClassLoader加载

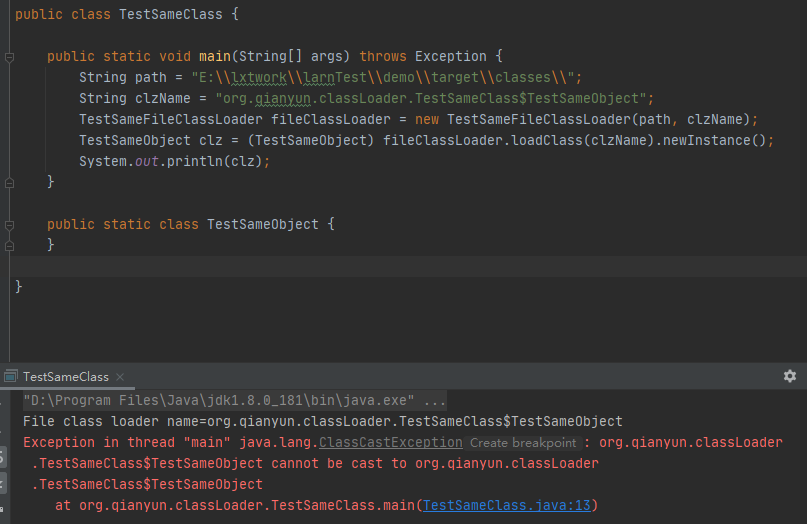


* 虽然是两个不同的ClassLoader加载，但是加载到是相同的Class，所以能强转成功
* 这时需要破坏双亲委派模型

## 破坏双亲委派自定义ClassLoader



* 运行报错



## 面向抽象类/接口 Api方式热更新调用

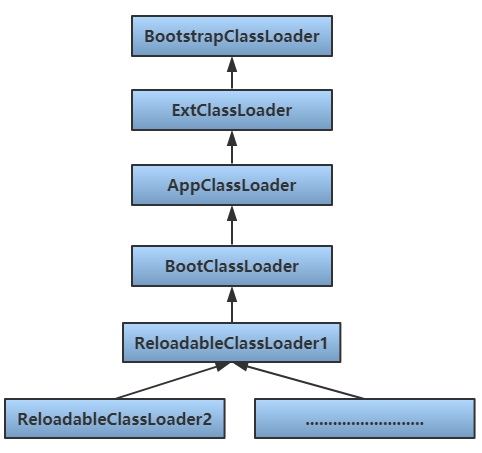
* 综上所述，面向实现进行热更新调用，将会抛出ClassCastException
* 我们需要转换成 A a = (A)new ClassLoader().loaderClass(“B”).newInstance()
* 只能面向抽象类/接口，有B继承A
* 则能以 A a = (A)new ClassLoader().loaderClass(“B”).newInstance(),这种方式达成热更
* 这个也是我们游戏里面使用的方式

# 游戏自定义ClassLoader问题

## 全部错误信息

* java.lang.IllegalAccessError
* tried to access class lilith.server.tank.game.arena.ArenaUtilImpl$1
* from class lilith.server.tank.game.arena.ArenaUtilImpl

## 游戏内热更新ClassLoader结构图

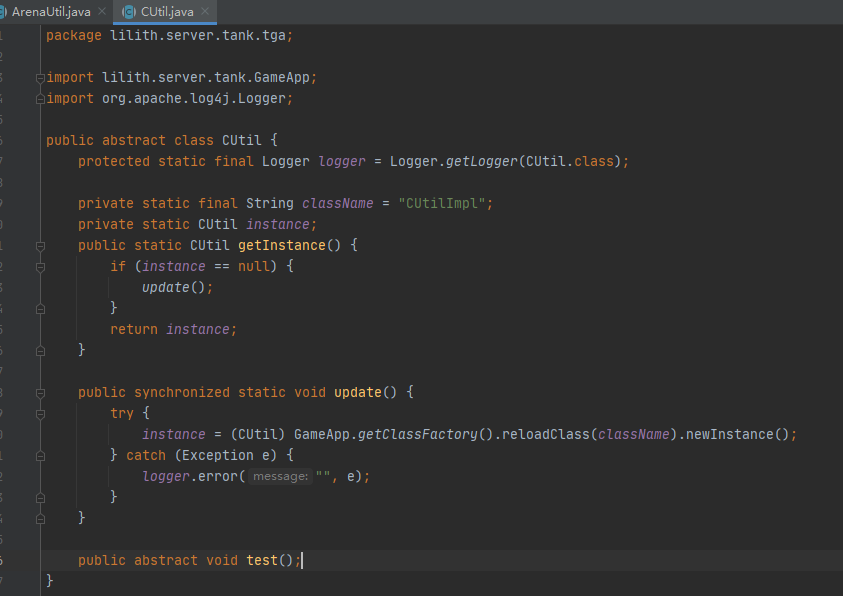


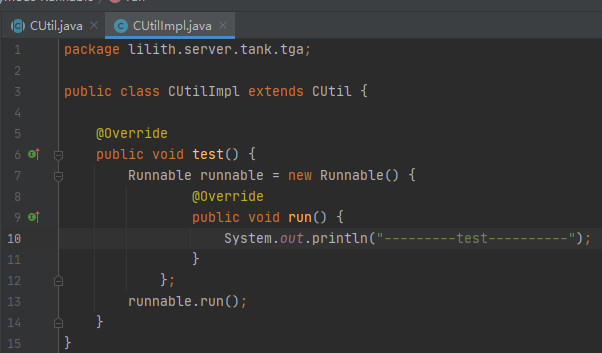
## 游戏内热更新大概流程

* 调用ClassFactory.reloadClass，设热更新class为className
* 移除已经加载的className对应的Class对象
* 并且将className添加到热更新列表里
* 执行ClassFactory.loadClass函数
* 判断className是否在热更新列表里面
  + 否- 返回父类加载器加载
  + 是- 继续执行
* 已经加载Class列表里面是否有 className的Class
  + 有- 直接返回
  + 没有- 继续执行
* 设当前的ReloadableClassLoader为curLoader, curLoader是否加载过对应的类
  + 是- 重新new，覆盖curLoader
  + 否- 继续执行
* 从配置路径加载className对应的二进制数据
* 将className，添加到curLoader 热更新列表
* 执行curLoader .loadClass逻辑
* 是否在热更新列表里面
  + 是- 执行破坏双亲委派加载class逻辑
  + 否- 回调ClassFactory.loadClass，回到ClassFactory.loadClass那一步
* 返回加载的class

## 出异常流程

* 在进行reloadClass时，先移除已加载的Class
* 同时移除curLoader 热更新列表 里面对应的className
* 设有线程T1 / T2，待热加载的Class为CUtilImpl,abstract class CUtil
* 则按照游戏内热更新模板代码如下图

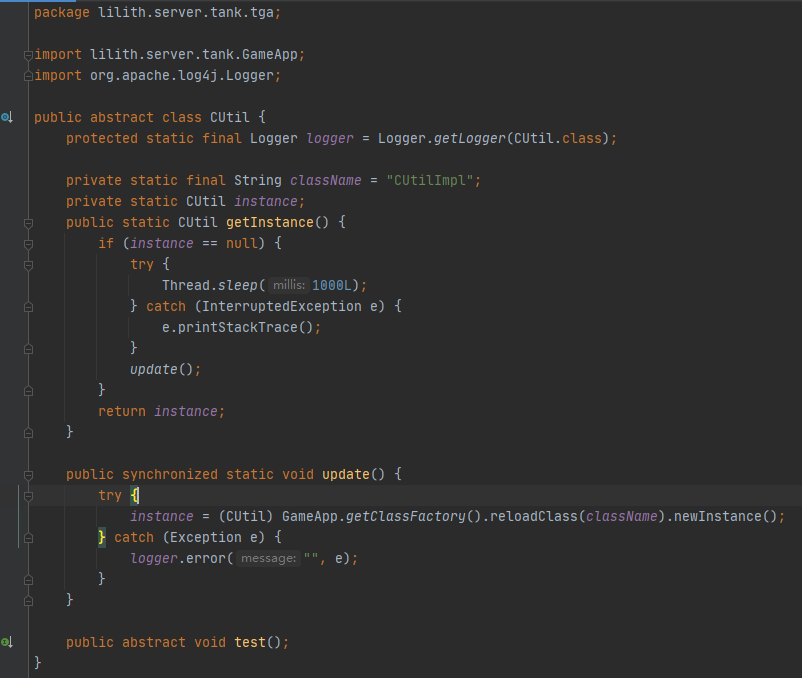


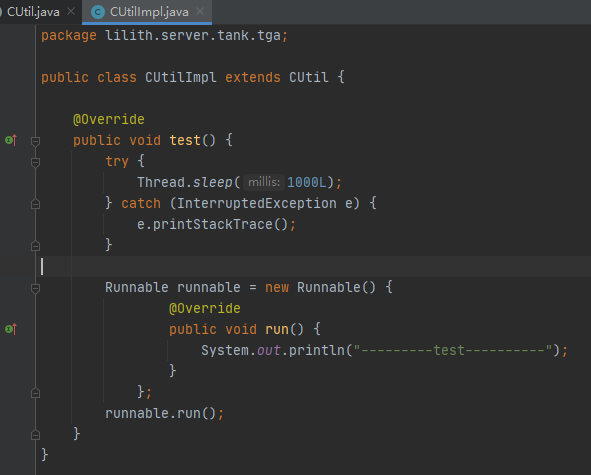


* 假设T1/T2 开始执行 getInstance
* 同时进入if代码块
* T1先执行update，使用CL1加载CUtilImpl
* T1执行test，T1线程失去CPU时间片
* T2执行update，CL1热更新列表移除CUtilImpl
* CL1已经加载过CUtilImpl，重新实例化CL2
* T2执行test函数，T2失去CPU时间片
* T1继续执行test函数
* 使用CL1加载CUtilImpl$1
* 由于CL1热更新列表已经移除CUtilImpl相关类
* 破坏双亲委派加载逻辑失效，重新回到ClassFactory.loadClass，使用CL2进行加载
* 导致CUtilImpl当前实例的类加载器与CUtilImpl$1加载器不是同一个，从而报错

## 如何必定出现该错误

* 分别在getInstance/test函数下加入sleep代码，如下图





# 参考资料

* 深入Java虚拟机(第二版)
  + 第3.3 类装载体系结构
  + 第8 动态连接和解析
* 深入理解Java虚拟机 JVM高级特性与最佳实践 第7章 虚拟机类加载机制
* [详细深入分析 Java ClassLoader 工作机制](https://segmentfault.com/a/1190000008491597)