

```

/*
    当前类加载器 (Current Classloader)

    每个类都会使用自己的类加载器 (即加载自身的类加载器) 来去加载其他类 (指的是所依赖的类),
    如果ClassX引用了ClassY, 那么ClassX的类加载器就会去加载ClassY (前提是ClassY尚未被加载)

    线程上下文类加载器 (Context Classloader)

    线程上下文类加载器是从JDK 1.2开始引入的, 类Thread中的getContextClassLoader()与setContextClassLoader(ClassLoader cl)
    分别用来获取和设置上下文类加载器。

    如果没有通过setContextClassLoader(ClassLoader cl)进行设置的话, 线程将继承其父线程的上下文类加载器。
    Java应用运行时的初始线程的上下文类加载器是系统类加载器。在线程中运行的代码可以通过该类加载器来加载类与资源。

    线程上下文类加载器的重要性:

    SPI (Service Provider Interface)

    父ClassLoader可以使用当前线程Thread.currentThread().getContextClassLoader()所指定的classloader加载的类。
    这就改变了父ClassLoader不能使用子ClassLoader或是其他没有直接父子关系的ClassLoader加载的类的情况, 即改变了
    双亲委托模型。

    线程上下文类加载器就是当前线程的Current Classloader。

    在双亲委托模型下, 类加载是由下至上的, 即下层的类加载器会委托上层进行加载。但是对于SPI来说, 有些接口是Java核心库所提供的,
    而Java核心库是由启动类加载器来加载的, 而这些接口的实现却来自于不同的jar包 (厂商提供), Java的启动类加载器是不会加载其他
    来源的jar包, 这样传统的双亲委托模型就无法满足SPI的要求。而通过给当前线程设置上下文类加载器, 就可以由设置的上下文类加载器
    来实现对于接口实现类的加载。

```

package main.java.Test1;

```

/*
    线程上下文类加载器的一般使用模式(获取--使用--还原)
    ClassLoader classLoader=Thread.currentThread().getContextClassLoader();
    try{
        Thread.currentThread().setContextClassLoader(targetTcc1);
        myMethod();
    }catch{
    }finally{
        Thread.currentThread().setContextClassLoader(classLoader);
    }

```

myMethod里面调用了Thread.currentThread().getContextClassLoader(),获取当前线程的上下文类加载器做一些事情

如果一个类由加载器A加载,那么这个类的依赖类也是由相同的类加载器加载(如果该依赖类之前没有被加载过得话)

ContextClassLoader的作用就是为了破坏Java类加载双亲委托机制

当高层提供统一的接口让低层去实现,同时又要在高层加载(或实例化)低层类的时候,就必须通过线程上下文类加载器

来帮助高层的classLoader找到并加载该类

```

*/
import java.sql.Driver;
import java.util.Iterator;
import java.util.ServiceLoader;

public class TestContextClassLoader {
    public static void main(String[] args) {
        //获取所有这个接口的实例
        ServiceLoader<Driver> loader=ServiceLoader.load(Driver.class);
        Iterator<Driver> iterator=loader.iterator();
        while(iterator.hasNext()){

```

```
        Driver driver=iterator.next();

        System.out.println("Driver:"+driver.getClass()+"",loader:"+driver.getClass().getClassLoader());

    }
    System.out.println("当前线程上下文类加载
器:"+Thread.currentThread().getContextClassLoader());
    System.out.println("ServiceLoader的类加载器:"+loader.getClass().getClassLoader());
}
}
```