

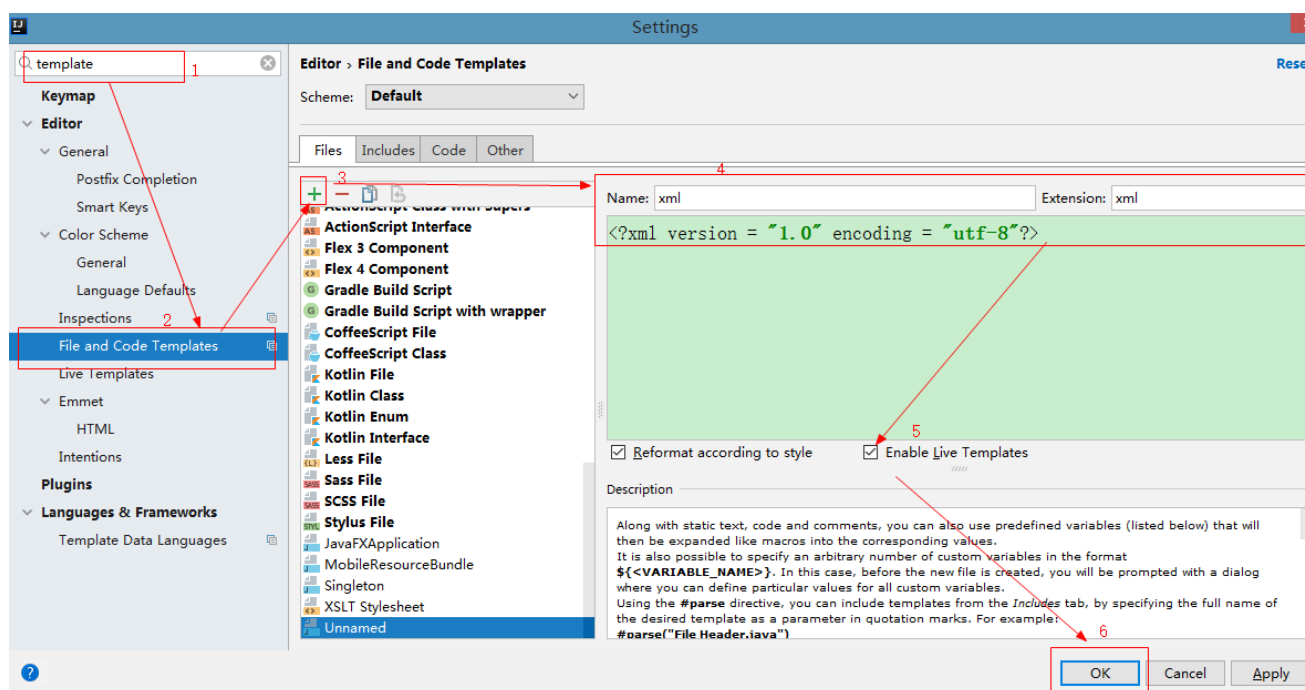
day26-xml和动态代理

一,XML入门【熟悉】

1.xml概述 可扩展标记语言, 标签可以自定义的

2.xml和html区别

定义xml模板



- html所有标签都是预定义的, xml所有标签都是自定义的
- html语法松散, xml语法严格, 区分大小写
- html做页面展示, xml描述数据

3.xml作用

- 作为配置文件。
- 存储数据。
- 用来传输数据。

4.xml语法规则

4.1文档声明

```
<?xml version = "1.0" encoding = "utf-8" standalone="yes" ?>
```

- 必须写在xml文档的第一行第一列
- 写法：<?xml version="1.0" ?>
- 属性:

1. version：版本号 固定值 1.0
2. encoding:指定文档的码表。默认值为 iso-8859-1
3. standalone：指定文档是否独立 yes 或 no(了解)

4.2元素：xml文档中的标签

- 文档中必须有且只能有一个根元素,其他标签都是这个根标签的子标签或孙标签
- 元素需要正确闭合
 1. 包含标签主体：somecontent
 2. 不含标签主体：
- 元素需要正确嵌套,不允许有交叉嵌套。
- 元素名称要遵守
 1. 元素名称区分大小写
 2. 数字不能开头
 3. 标签之间不能有空格,特殊字符

4.3属性

- 属性值必须用引号引起来。单双引号都行
- 在一个标签里面属性不能重复

4.4注释

语法

快捷键: Ctrl+Shift+/

- 文档声明前面不能有注释
- 注释不能嵌套

4.5特殊字符和CDATA区

1.在 XML 中有 5 个预定义的实体引用：

<	<	小于
>	>	大于
&	&	和号
'	'	省略号
"	"	引号

2.CDATA 内部的所有东西都会被解析器忽略,当做文本

语法:<![CDATA[内容]]>

- 本地DTD

```
<!DOCTYPE 根元素 SYSTEM "文件名">
```

- 网络DTD

```
<!DOCTYPE 根元素 PUBLIC "DTD名称" "DTD文档的URL">
```

2.1.2语法

1. 元素：<!ELEMENT 元素名称 元素组成>

- 元素组成：EMPTY，ANY，(子元素)，(#PCDATA)
- 如果出现子元素: ?(出现0次或1次),
*(出现0次或多次),
+(出现1次或多次) 表示子元素出现的个数.
|或者（只能出现一个）
,代表子元素出现必须按照顺序

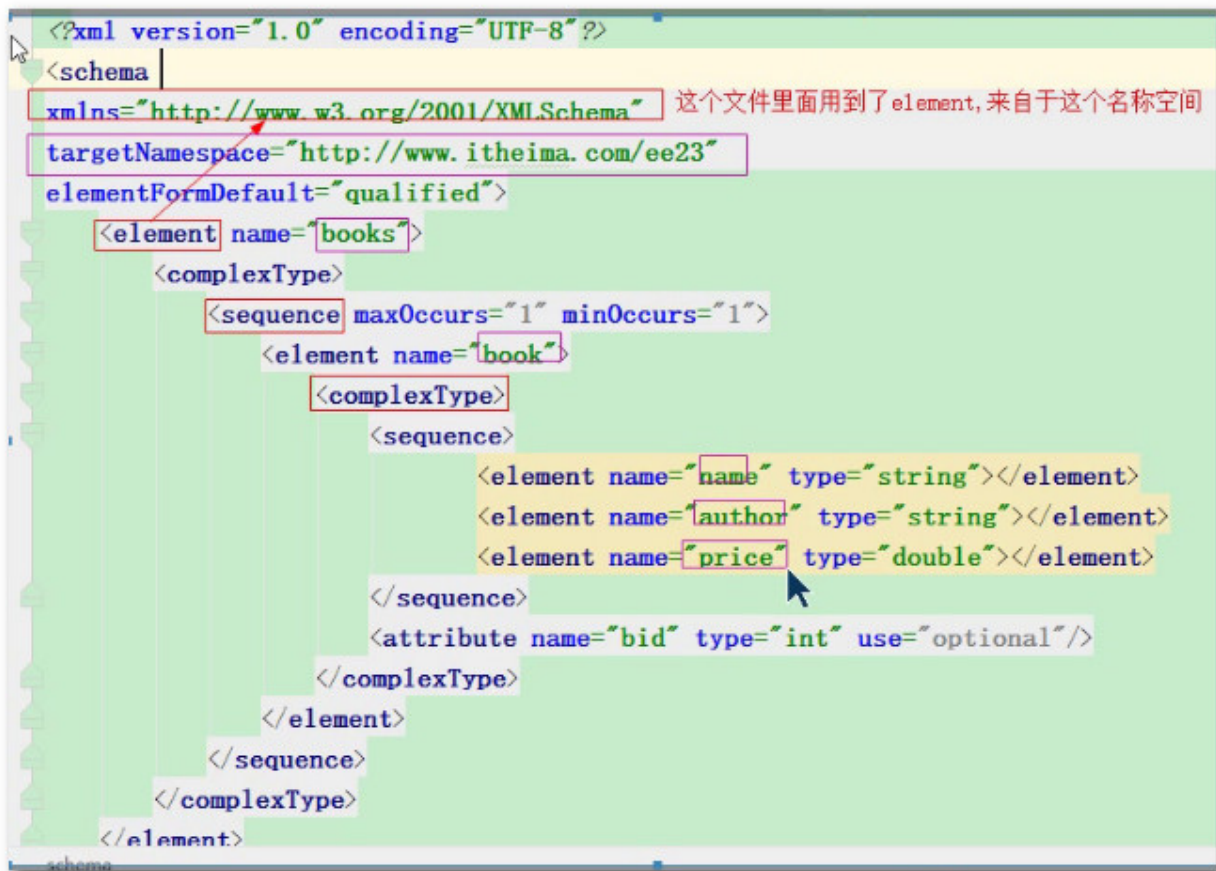
2. 属性<!ATTLIST 元素名称 属性名称 属性类型 属性使用规则>

- 属性类型: ID,枚举,CDATA
- 使用规则: #REQUIRED:属性值是必需的
#IMPLIED :属性不是必需的
#FIXED value:属性值是固定的
ID使用：字母开头

2.2schema

2.2.1 dtd和schema的区别

- XMLSchema符合XML语法结构。
- DOM、SAX等XMLAPI很容易解析出XML Schema文档中的内容。
- XMLSchema对名称空间支持得非常好。名称空间可以理解为前缀
- XMLSchema比XML DTD支持更多的数据类型，并支持用户自定义新的数据类型。
- XMLSchema定义约束的能力非常强大，可以对XML实例文档作出细致的语义限制。
- 但Xml Schema现在已是w3c组织的标准，它正逐步取代DTD。
- XMLSchema不能像DTD一样定义实体，比DTD更复杂，



2.2.1根据schema约束写xml文件步骤

- 查出Schema文档,找出根元素
- 根元素来自哪个名称空间
- 这个名称空间和哪个xsd文件对应(指定约束的路径)
- schemaLocation不是关键字, 来自一个标准的名称空间

book.xml代码示例

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- 引入W3C规定的实例文档的名称空间. -->
<books
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns="http://www.itheima.com/ee24"
  xsi:schemaLocation="http://www.itheima.com/ee24 book.xsd">
  <book bid="1">
    <name>Java编程思想</name>
    <author>James</author>
    <price>100</price>
  </book>
</books>
```

book.xsd代码示例

```
<?xml version="1.0" encoding="UTF-8"?>
```

```
<schema
  xmlns="http://www.w3.org/2001/XMLSchema"
  targetNamespace="http://www.itheima.com/ee24"
  elementFormDefault="qualified">
  <element name="books">
  <complexType>
  <sequence maxOccurs="1" minOccurs="1">
  <element name="book">
    <complexType>
    <sequence>
      <element name="name" type="string"></element>
      <element name="author" type="string"></element>
      <element name="price" type="double"></element>
    </sequence>
    <attribute name="bid" type="int" use="required"/>
    </complexType>
  </element>
  </sequence>
  </complexType>
</element>
</schema>
```

三,XML解析【掌握】

1.xml解析方式 两种思想

1.1DOM解析 全部加载到内存

将xml文档加载到内存，形成一颗dom树(document对象)，将文档的各个组成部封装为一些对象。

优点: 因为，在内存中会形成dom树，可以对dom树进行增删改查。

缺点: 如果xml文件太大, dom树非常占内存，解析速度慢。

Document:文档节点

Element:元素(标签)节点

Text:文本节点

Attribute:属性节点

Comment:注释节点

1.2SAX解析 一次读一行，读完内存释放


逐行读取，基于事件(函数或者方法)驱动

优点：内存占用很小，速度快

缺点：只能读取，不能回写

```
<?xml version="1.0" encoding="UTF-8" ?>

<students>
  <student>
    <name>zs</name>
    <age>abc</age>
    <sex>yao</sex>
  </student>
</students>
```


读取器

startDocument : 开始文档
startElement : 开始元素
characters:解析文本
endElement:结束元素
endDocument : 结束文档

2.常用xml解析器(jar包)

JAXP : sun公司提供的解析。支持dom和sax。

JDOM

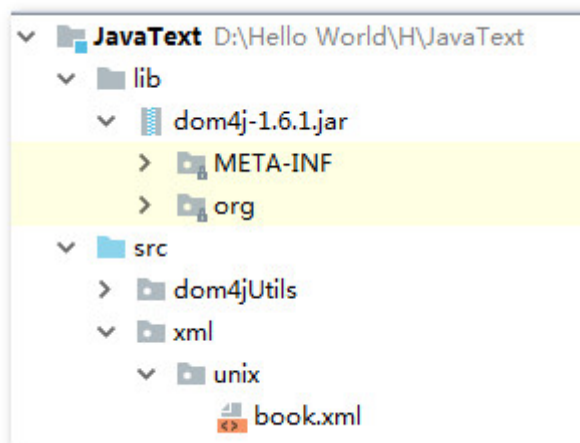
DOM4J : document for java 民间方式 , 但是是事实方式。非常好。 支持dom和sax.

JSOUP: 爬虫, 解析html

3.Dom4J的使用【重点】

3.1使用步骤

1. 导入jar包 dom4j-1.6.1j.jar,准备xml文件,xml必须放在src目录下否则无法识别



```
<?xml version="1.0" encoding="UTF-8"?>
<books>
  <book id="b1">
    <name>四十二章经</name>
    <autor>不详</autor>
    <price>10000</price>
  </book>
  <book id="b2">
    <name>葵花宝典</name>
```

```
<autor>小李子</autor>
<price>8888</price>
</book>
<apple>
  <name>苹果</name>
</apple>
</books>
```

2.java中创建解析器

```
SAXReader sax = new SAXReader();
```

使用类加载器读取xml文件得到 InputStream

```
InputStream is = Dom4jUtils.class.getClassLoader().getResourceAsStream("xml/unix/book.xml");
```

使用SAXReader的read方法读取流得到Document对象

```
Document d=sax.read(is);
```

得到目录最上级根元素

```
Element rootele = d.getRootElement();
```

3.2常见的API

- 获取所有子元素节点

```
List<Element> list=element.elements();
```

- 获取节点属性的值

```
element.attributeValue("属性名")或者element.attribute("属性名").getValue();
```

- 获得节点的文本值

```
element.getText();
```

3.3示例代码

- Dom4j获取孙标签内的名字

```
public class Dom4jUtils {
```



```

    public static void main(String[] args) throws DocumentException {
        SAXReader sax=new SAXReader();
        InputStream is =
Dom4jUtils.class.getClassLoader().getResourceAsStream("xml/unix/book.xml");
        Document d=sax.read(is);
        Element rootele = d.getRootElement();
        List<Element> list=rootele.elements();
        Element book1 = list.get(0);
        List<Element> list2=book1.elements();
        Element book2=list2.get(0);
        String name=book2.getText();
        System.out.println(name);
    }
}

```

- Dom4j获取子标签属性值

```

public class Dom4jUtils {
    public static void main(String[] args) throws DocumentException {
        SAXReader sax=new SAXReader();
        InputStream is =
Dom4jUtils.class.getClassLoader().getResourceAsStream("xml/unix/book.xml");
        Document d=sax.read(is);
        Element rootele = d.getRootElement();
        List<Element> list=rootele.elements();
        Element book1 = list.get(0);
        String id=book1.attributeValue("id");
        System.out.println(id);
    }
}

```

4.XPath的使用,基于dom4j XPath在xml有约束的情况下无法使用！

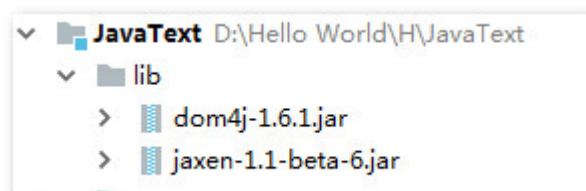
4.1介绍

Xpath定义了一种规则,专门用于查询xml, 可以认为xpath是dom4j的扩展

Xpath不需要获得根元素，直接用Document对象找

4.2使用步骤

1. 导入jar包 jaxen...jar(当前资料里是: jaxen-1.1-beta-6.jar) 注意: 两个(dom4j的jar包和xpath的jar包)



1. 创建解析器

```
SAXReader reader = new SAXReader();
```

2. 使用类加载器读取xml文件得到 InputStream

```
InputStream is =  
Dom4jUtils.class.getClassLoader().getResourceAsStream("xml/unix/book.xml");
```

3. 解析xml 流获得document对象

```
Document document = reader.read(is);
```

4.3常见的API

- 获得一个节点(标签,元素)

```
document.selectSingleNode("xpath语法");
```

- 获得多个节点(标签,元素)

```
document.selectNodes("xpath语法");
```

Xpath文档语法

The screenshot shows the XPath Tutorial website. The left sidebar lists examples 1 through 10. The main content area shows the text of example 2: "如果路径以双斜线 // 开头, 则表示选择文档中所有满足双斜线//之后规则的元素(无论层级关系)". To the right, there is a tree view diagram illustrating the XPath expression `//BBB`. The diagram shows a root node `<AAA>` with three children: `<BBB/>`, `<CCC/>`, and `<DDD>`. The `<BBB/>` node is highlighted in blue, and a red arrow points to it with the text "对应关系" (Correspondence). Below the tree view, there is a link to "在XLab中打开实例 | 树视图 (JPG)".

4.4示例代码

xml文件

```
<?xml version="1.0" encoding="UTF-8"?>  
  
<books>
```

```
<book id="b1">
  <name>四十二章经</name>
  <autor>不详</autor>
  <price>10000</price>
</book>
<book id="b2">
  <name>葵花宝典</name>
  <autor>小李子</autor>
  <price>8888</price>
</book>
<apple>
  <name>苹果</name>
</apple>
</books>
```

获取四十二章经

```
public class XpathUtils {
    public static void main(String[] args) throws DocumentException {
        SAXReader sax=new SAXReader();
        InputStream
is=XpathUtils.class.getClassLoader().getResourceAsStream("xml/unix/book.xml");
        Document d=sax.read(is);
        Element ele= (Element) d.selectSingleNode("/books/book/name");
        System.out.println(ele.getText());
    }
}
```

获得第二本书

方块号里的表达式可以进一步的指定元素, 其中数字表示元素在选择集里的位置, 而last()函数则表示选择集中的最后一个元素.

/AAA/BBB[1]

选择AAA的第一个BBB子元素

```
<AAA>
  <BBB/>
  <BBB/>
  <BBB/>
  <BBB/>
</AAA>
```

[在XLab中打开实例 | 树视图 \(JPG\)](#)

/AAA/BBB[last()]

选择AAA的最后一个BBB子元素

```
<AAA>
  <BBB/>
  <BBB/>
  <BBB/>
  <BBB/>
</AAA>
```

[在XLab中打开实例 | 树视图 \(JPG\)](#)

```
public class XpathUtils {
    public static void main(String[] args) throws DocumentException {
        SAXReader sax=new SAXReader();
        InputStream
is=XpathUtils.class.getClassLoader().getResourceAsStream("xml/unix/book.xml");
        Document d=sax.read(is);
        Element ele= (Element) d.selectSingleNode("/books/book[2]");
        String name=((Element) ele.elements().get(0)).getText();
        System.out.println(name);
    }
}
```

四,动态代理

1.代理模式概述

1.1什么是代理模式

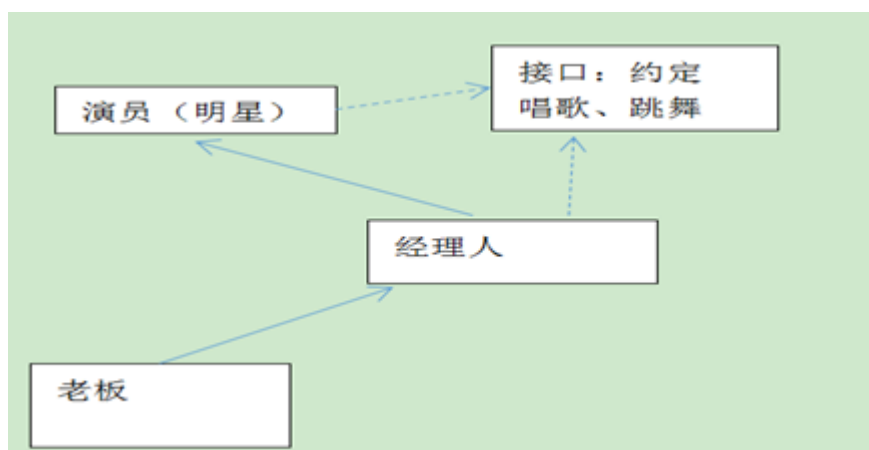
ProxyPattern（即：代理模式），23种常用的面向对象软件的设计模式之一

装饰者设计模式：在原有类的基础上提供一个接口,再另外创建一个类实现这个接口,利用构造对象传递，将原有的方法传递过来,同时增加一些方法,打到增强功能的目的。

代理模式：为其他对象提供一种代理以控制对这个对象的访问。在某些情况下，一个对象不适合或者不能直接引用另一个对象，而代理对象可以在客户端和目标对象之间起到**中介**的作用。

装饰者是利用对象传递增强方法，静态代理是由代理者new新对象实现增强,与原对象无关,动态代理是利用反射来增强原对象;

作用：增强一个类中的某个方法.对程序进行扩展. Spring框架中AOP.



静态代理的体现:

Car接口

```
public interface Car {
    void run();
    void stop();
}
```

原始QQ车

```

public class QQ implements Car{
    @Override
    public void run() {
        System.out.println("QQ跑60迈");
    }
    @Override
    public void stop() {
        System.out.println("QQ刹车");
    }
}

```

改装QQ车

```

public class WrapperCar implements Car{
    private Car car;
    public WrapperCar() {
        this.car = new QQ(); //重点:代理体现, 改由WrapperCar代替QQ来new对象
    }
    @Override
    public void run() {
        System.out.println("5秒破百");
    }
    @Override
    public void stop() {
        car.stop();
    }
}

```

测试类

```

public class Test {
    public static void main(String[] args) {
        WrapperCar wc=new WrapperCar();
        wc.run();
        wc.stop();
    }
}

```

1.2动态代理介绍

动态代理它可以直接给某一个实现了某些接口的被代理对象生成一个代理对象，而不需要代理类存在。如上例中，不需要WrapperCar这个类存在。

动态代理与代理模式原理是一样的，只是它没有具体的代理类，直接通过反射生成了一个代理对象。

- 动态代理的分类

jdk提供一个Proxy类可以直接给实现接口类的对象直接生成代理对象

spring中动态代理:cglib

2.jdk中的动态代理的使用

2.1API介绍

java.lang.reflect.Proxy类可以直接生成一个代理对象

- 生成一个代理对象

```
Proxy.newProxyInstance(ClassLoader loader, Class<?>[] interfaces, InvocationHandler h)
```

- 参数1: `ClassLoader loader` 被代理对象的类加载器
- 参数2: `Class<?>[] interfaces` 被代理对象实现的接口字节码对象数组

上例中可以写 `new Class[]{Car.class}` 或者 `qq.getClass().getInterface()`;

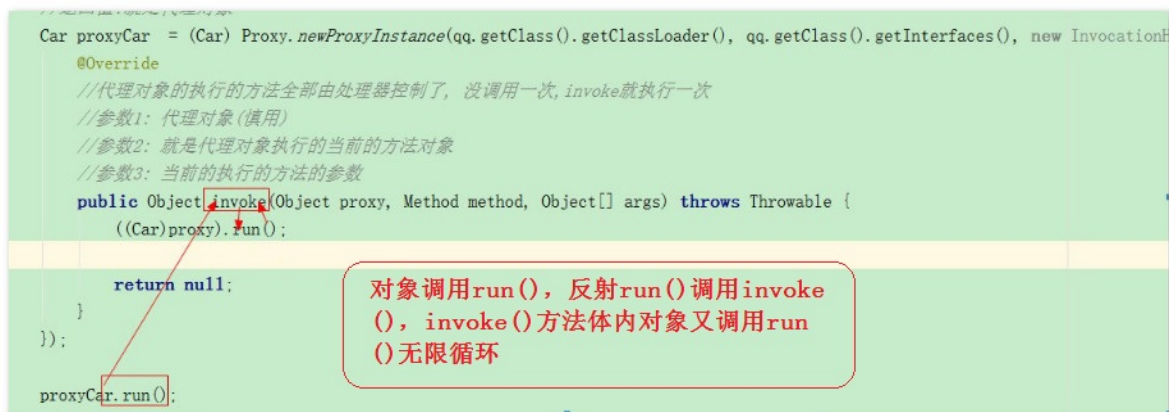
- 参数3: `InvocationHandler h` (接口) 执行处理类 代理对象, 方法逻辑全部由处理器控制。每调用一次 `invoke` 就执行一次, 匿名内部类如下

```
new InvocationHandler(){
    @Override
    public Object invoke(Object proxy, Method method, Object[] args) throws Throwable {
        //System.out.println(method.getName());
        return null;
    }
}
```

- `InvocationHandler` 中的 `invoke(Object proxy, Method method, Object[] args)` 方法：调用代理类的任何方法，此方法都会执行

- `Object proxy`: 代理对象

不要在方法体中用代理对象调用方法，会出现无限递归，栈内存溢出



原因：method通过字节码对象clazz获得，然后method.invoke(对象,参数)运行方法

所以每执行一次方法就调用一次invoke

```
Method method=clazz.getDeclaredMethod("speak1",String.class);
method.invoke(clazz.newInstance(),"李四");
```

- Method method:当前执行的方法
- Object[] args:当前执行的方法运行时传递过来的参数
 - 返回值:当前方法执行的返回值

2.2代码实现

动态代理QQ车改装案例

接口Car

```
public interface Car {  
    void run();  
    void stop();  
    String addOil(int num);  
}
```

原QQ类

```
public class QQ implements Car{  
    @Override  
    public void run() {  
        System.out.println("QQ跑60迈");  
    }  
    @Override  
    public void stop() {  
        System.out.println("QQ刹车");  
    }  
  
    @Override  
    public String addOil(int num) {  
        return "QQ"+num;  
    }  
}
```

动态代理类

```
public class ProxyUtils {  
    public static Object createProxy(Object o){ //传入被代理类对象，返回增强代理类对象  
        Car proxyqq= (Car)  
        Proxy.newProxyInstance(o.getClass().getClassLoader(),o.getClass().getInterfaces(),new  
        InvocationHandler(){  
            @Override  
            public Object invoke(Object proxy, Method method, Object[] args) throws Throwable {  
                if("run".equals(method.getName())){  
                    //如果是run()方法就增强,因为run本身是void方法,所以要返回null将原方法覆盖  
                    System.out.println("5秒破百");  
                    return null;  
                }  
                //如果不是run()方法，就不需要增强,直接调用  
                return method.invoke(o,args);  
            }  
        });  
    }  
}
```



```
        }  
    }  
    );  
    return proxyqq;  
}  
}
```

测试类

```
public class Test {  
    public static void main(String[] args) {  
        QQ qq=new QQ();  
        Car proxyqq= (Car) ProxyUtils.createProxy(qq);  
        String str=proxyqq.addOil(92);  
        System.out.println(str);  
        proxyqq.run();  
        proxyqq.stop();  
    }  
}
```