



Custom Search

Search

[Home](#) > [Core Java](#)

Dynamic Proxy with Proxy and InvocationHandler in Java

By Arvind Rai, March 31, 2014

A proxy calls object method indirectly through the proxy object. java.lang.reflect API provides a class as Proxy and an interface as InvocationHandler. Together these two API creates dynamic proxy class. Proxy class creates the dynamic proxy class on the basis of given parameters. InvocationHandler invokes the methods of dynamic proxy class. All these happening will be discussed below in detail with an example.

Proxy Class in Java

java.lang.reflect.Proxy is a class that provides static methods to create Dynamic Proxy class. There is a method as newProxyInstance() inside Proxy class which is defined as below.

```
public static Object newProxyInstance(ClassLoader loader, Class<?>[] interfaces, InvocationHandler h)
```

To understand the arguments, I will take the example that I have an interface names as Task and its implementation class is TaskImpl. *Dynamic proxy class will be created of TaskImpl class.* Now understand how the argument will be passed in <>Proxy.newProxyInstance() method.

1. ClassLoader : This class loader will define the dynamic proxy class. Class loader can be obtained by class or interface whose dynamic proxy is being created. If we try to get it by interface, call it as below

```
Task.class.getClassLoader()
```

and if we have class instance as ob of TaskImpl then class loader will be obtained as

```
ob.getClassLoader()
```

2. The second argument needs the array of all the interfaces which will be implemented by dynamic proxy class.
3. Pass the instance of class which is implementing java.lang.reflect.InvocationHandler.

Find the properties of dynamic proxy class.

1. A proxy class is public and final. It cannot be an abstract class.
2. Each proxy class extends java.lang.reflect.Proxy.
3. If there is non-public interface in the list passed to create dynamic proxy, then that interface must be in the same package otherwise it will not be accessible
4. Each proxy class has one public constructor that takes one argument. By this constructor InvocationHandler instance is set.
5. Every proxy class will be associated with a InvocationHandler.



Treasure Data Recognized in the 2018 Gartner Magic Quadrant for Data Management Solutions for Analytics



InvocationHandler in Java

InvocationHandler is an interface in java.lang.reflect package. InvocationHandler is implemented by a user class to invoke method of dynamic proxy class. The syntax of invoke method is as below.

```
Object invoke(Object proxy, Method m, Object[] args)
```

The description of arguments is given below.

Object : This is the proxy instance on which method is invoked.




Method: This corresponds to interface method which is invoked on proxy instance.

Object[]: It contains an array of arguments passed in method invocation.

THE 2018 JEEP COMPAS
Get the Best-in-Class highway fuel economy of 32 MPG¹

Legal

BUILD & I
SEARCH

Latest Post

Angular Module Loading: Eager, Lazy and Preloading

Jackson @JsonIgnore, @JsonIgnoreProperties and @JsonIgnoreType

Jackson @JsonAnyGetter and @JsonAnySetter Example

Jackson @JsonProperty and @JsonAlias Example

Jackson @JacksonInject Example

Example to Create Dynamic Proxy Class

Now we will discuss the example to create the dynamic proxy class.
This is the sample interface.

Task.java

```
package com.concretepage.proxy;

public interface Task {

    public void setData(String data);

    public int getCalData(int x);

}
```

The sample implementation of the interface is given below.

TaskImpl.java

```
package com.concretepage.proxy;

public class TaskImpl implements Task {

    @Override
    public void setData(String data) {
        System.out.println(data+ " Data is saved");
    }

    @Override
    public int getCalData(int x) {
        return x*10;
    }

}
```

Find the InvocationHandler implementation. invoke() method has be defined.

MyInvocationHandler.java

```
package com.concretepage.proxy;

import java.lang.reflect.InvocationHandler;
import java.lang.reflect.InvocationTargetException;
import java.lang.reflect.Method;

public class MyInvocationHandler implements InvocationHandler{

    private Object obj;

    public MyInvocationHandler(Object obj) {
        this.obj = obj;
    }

    public Object invoke(Object proxy, Method m, Object[] args) throws Throwable {
        Object result;
        try{
            if(m.getName().indexOf("get")>-1){
                System.out.println("...get Method Executing...");
            }else{
                System.out.println("...set Method Executing...");
            }
            result = m.invoke(obj, args);
        } catch (InvocationTargetException e) {
            throw e;
        } catch (Exception e) {
            throw e;
        }
        return result;
    }

}
```

Create a factory class to get dynamic proxy class.

```
package com.concretepage.proxy;

import java.lang.reflect.Proxy;

public class ProxyFactory {

    public static Object newInstance(Object ob) {
        return Proxy.newProxyInstance(ob.getClass().getClassLoader(),
```

Top Trends
Angular 2 Http post() Example
Angular 2 Radio Button and Checkbox Example
Angular 2 Http get() Parameters + Headers + URLSearchParams + RequestOptions Example
Java 8 Stream: allMatch, anyMatch and noneMatch Example
Spring Boot REST + JPA + Hibernate + MySQL Example



Popular Post
Angular 2 Select Option + Multiple Select Option + Validation Example using Template-Driven Form

```
        new Class<?>[] { Task.class }, new MyInvocationHandler(ob));
    }
}
```

Now this is the time to test our dynamic proxy class by calling its method.
Test.java

```
package com.concretepage.proxy;
public class Test {
    public static void main(String[] args) {
        Task task = (Task)ProxyFactory.newInstance(new TaskImpl());
        task.setData("Test");
        System.out.println(task.getCalData(5));
    }
}
```

Find the output. **Output**

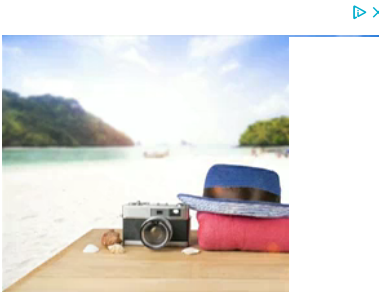
```
...set Method Executing...
Test Data is saved
...get Method Executing...
50
```

What if when Proxy Interfaces have Multiple Duplicated Methods?

To create dynamic proxy class, we need to pass an array of all those interfaces which will be implemented by our dynamic class. Suppose we pass two interfaces having duplicate methods. These duplicate methods will be same in arguments but will differ in return type. Now what will be of those duplicate methods while invoking methods of the dynamic proxy class? Here the order of those methods will be significant and foremost interface method will be invoked.

Share

Tweet



POSTED BY

ARVIND RAI

Popular Tutorials:

Java 8

Spring 4

Angular

Struts 2

Android

FIND MORE TUTORILAS

Angular 2 NgIf Example
Java 8 Stream sorted() Example
Angular 4 CRUD Example
Angular 2 @ViewChild() Example

Featured Post
Angular 4 NgIf-Then-Else Example
Angular HttpClient get Example
Java 8 Stream reduce() Example
Angular 2/4 Email Validation Example
Angular 2 Decimal Pipe, Percent Pipe and Currency Pipe Example

第3页 共4页

2018/3/22 下午4:59

Java Callable Example	Example of Observer and Observable in Java
ServiceLoader Java Example	Angular 4 CRUD Example
MyBatis 3 Annotation Example with @Select, @Insert, @Update and @Delete	RecursiveTask Java Example
Example of defaultThreadFactory in Java	How to write generic method in java Generic method Example
Example of Singleton in Java	Example of Serialization and Deserial in Java
Java 8 Stream reduce() Example	Java 8 Supplier Example
Spring Boot CrudRepository Example	Example of AtomicMarkableReferenc Java
Example of ManagedBlocker in Java	Example Of CountdownLatch in Java

SPRING BOOT


HIBERNATE

PRIMEFACES

RESTEASY

FREEMARKER

Login ↗



Leave your comment...

CommentsSort by Newest

Be the first to comment!

ADD WIDGETPACK TO YOUR SITE

POWERED BY WIDGET PACK™

Favorite Links

Java Technology

Hibernate Annotations

Spring Framework

JQuery

Apache Struts

MyBatis

Quartz Scheduler

Angular

Thymeleaf

FreeMarker

About Us

We are a group of software developers.


We enjoy learning and sharing technologies.


To improve the site's content,


your valuable suggestions

are most welcome. *Thanks*

Email : concretepage@gmail.com








Mobile Apps

ConcretePage.com



SCJP Quiz

