

Dynamic Proxy

Java Collections
2 уровень, 7 лекция

ОТКРЫТА

— Привет, Амиго.

— Здорово, Риша.

— Сегодня я расскажу тебе новую и очень интересную тему — **динамические прокси**.

В Java есть несколько способов изменить функциональность нужного класса...

Способ первый. Наследование

Самый простой способ изменить поведение некоторого класса — это создать новый класс, унаследовать его от оригинального (базового) и переопределить его методы. Затем, вместо объектов оригинального класса использовать объекты класса наследника. Пример:

1	Reader reader = new UserCustomReader();
---	---

Способ второй. Использование класса-обертки (Wrapper).

Примером такого класса является **BufferedReader**. Во-первых, он унаследован от **Reader**, то есть может быть использован вместо него. Во-вторых, он переадресует все вызовы к оригинальному объекту **Reader**, который обязательно нужно передать в **конструкторе** объекту **BufferedReader**. Пример:

1	Reader readerOriginal = new UserCustomReader();
2	Reader reader = new BufferedReader(readerOriginal);

Способ третий. Создание динамического прокси (Proxy).



В Java есть специальный класс (java.lang.reflect.Proxy), с помощью которого фактически **можно сконструировать объект во время исполнения программы (динамически), не создавая для него отдельного класса.**

Это делается очень просто:

1	Reader reader = (Reader)Proxy. newProxyInstance ();
---	--

— А вот это уже что-то новенькое!

— Но, нам ведь не нужен просто объект без методов. Надо чтобы у этого объекта были методы, и они делали то, что нам нужно. Для этого в Java используется специальный интерфейс **InvocationHandler**, с помощью которого **можно перехватывать все вызовы методов**, обращенные к проху-объекту. проху-объект можно создать только используя интерфейсы.

Invoke – стандартное название для метода/класса, основная задача которого просто вызвать какой-то метод.

Handler – стандартное название для класса, который обрабатывает какое-то событие. Например, обработчик клика мышки будет называться MouseClickHandler, и т.д.

У интерфейса InvocationHandler есть единственный **метод invoke, в который направляются все вызовы, обращенные к проху-объекту**. Пример:

Код	
1	Reader reader = (Reader)Proxy. newProxyInstance (new CustomInvocationHandler());
2	reader.close ();

1	class CustomInvocationHandler implements InvocationHandler
2	{
3	public Object invoke (Object proxy, Method method, Object[] args) throws Throwable
4	{
5	System.out.println ("yes!");
6	return null ;
7	}
8	}

При вызове метода **reader.close()**, вызовется метод **invoke**, и на экран будет выведена надпись “yes!”

— Т.е. мы объявили класс **CustomInvocationHandler**, в нем реализовали интерфейс **InvocationHandler** и его метод **invoke**. Метод **invoke** при вызове выводит на экран строку “yes!”- Затем мы создали объект типа **CustomInvocationHandler** и передали его в метод **newProxyInstance** при создании объекта-проху.

— Да, все верно.

Это очень мощный инструмент, обычно создание таких прокси используется для имитации **объектов из программ, которые физически запущены на другом компьютере**. Или для контроля доступа

– в таком методе можно проверять права текущего пользователя, обрабатывать ошибки, логировать ошибки и многое другое.

Вот пример, где метод **invoke** еще и вызывает методы оригинального объекта:

Код	
1	Reader original = new UserCustomReader();
2	
3	Reader reader = (Reader)Proxy. newProxyInstance (new CustomInvocationHandler(original));
4	reader.close ();

```
1  class CustomInvocationHandler implements InvocationHandler
2  {
3      private Reader readerOriginal;
4
5      CustomInvocationHandler(Reader readerOriginal)
6      {
7          this.readerOriginal = readerOriginal;
8      }
9
10     public Object invoke(Object proxy, Method method, Object[] args) throws Throwable
11     {
12         if (method.getName().equals("close"))
13         {
14             System.out.println("Reader closed!");
15         }
16
17         // это вызов метода close у объекта readerOriginal
18         // имя метода и описание его параметров хранится в переменной method
19         return method.invoke(readerOriginal, args);
20     }
21 }
```

В данном примере есть две особенности.

Во-первых, в конструктор передается «оригинальный» объект **Reader**, ссылка на который сохраняется внутри **CustomInvocationHandler**.

Во-вторых, в методе `invoke` мы снова вызываем этот же метод, но уже у «оригинального» объекта.

— Ага. Т.е. вот эта последняя строчка и есть вызов того же самого метода, но уже у оригинального объекта:

```
1  return method.invoke(readerOriginal, args);
```

— Ага.

— Не сказал бы, что слишком очевидно, но все же понятно. Вроде бы.

— Отлично. Тогда вот еще что. В метод `newProxyInstance` нужно передавать еще немного служебной информации для создания проху-объекта. Но, т.к. мы не создаем монструозные прокси-объекты, то эту информацию легко получить из самого оригинального класса.

Вот тебе пример:

Код	
1	Reader original = new UserCustomReader();
2	
3	ClassLoader classLoader = original.getClass().getClassLoader();
4	Class<?>[] interfaces = original.getClass().getInterfaces();
5	CustomInvocationHandler invocationHandler = new CustomInvocationHandler(original);
6	
7	Reader reader = (Reader)Proxy.newProxyInstance(classLoader, interfaces, invocationHandler);

```
1  class CustomInvocationHandler implements InvocationHandler
2  {
```



```
5     return null;
6 }
7 }
```

— Ага. ClassLoader и список интерфейсов. Это что-то из Reflection, да?

— Ага.

— Ясно. Что ж, думаю, я смогу создать примитивный простенький прокси объект, если это когда-нибудь мне понадобится.

— А ты спроси у Диего

− +66 +

Комментарии (105)

популярныеновыестарые

JavaCoder

Введите текст комментария

ZlataУровень 49, Олыштын, Польша22 апреля, 09:20⋮



Ответить

− 0 +

Дмитрий ЩебрюкУровень 23, Москва, Russian Federation26 июня, 11:10⋮

Здесь что-то на программистком, не могу прочесть это.

Ответить

− 0 +

Фарид ГулиевУровень 41, Днепр, Украина2 июля, 16:32⋮

Да вроде Мордорский

Ответить

− 0 +

ДмитрийУровень 3823 июля, 19:34⋮

Кодировка непонятная.

Ответить

− 0 +

Ответить

Рогов Игорь

Уровень 50, Самара, Russian Federation

7 апреля, 12:57

...

жонглирование классами и объектами. следите за руками

Ответить

-

+1

+

Anna Avilova

architect

15 апреля, 17:38

...

в предыдущей лекции была ссылка на лекцию профессора о прокси. почитайте, станет понятнее.

Ответить

-

0

+

PaiMei in J#

Grand Master в Eagles' Claw

14 октября 2021, 17:44

...

"Полезные ссылки от профессора" с прошлого уровня

Ответить

-

+12

+

Жора Нет

Уровень 39, енакиево, Украина

7 апреля, 21:55

...

Ссылка - бомба!!!!!! По ней первая задача решается как по шаблону.

Ответить

-

0

+

Ян

Уровень 41, Лида, Беларусь

26 сентября 2021, 16:02

...

в последнем примере создаётся объект **CustomInvocationHandler** и в конструктор ему передаётся **Reader** original

1

CustomInvocationHandler invocationHandler = new CustomInvocationHandler(original);

но в самом классе **CustomInvocationHandler** только конструктор по умолчанию

1

class CustomInvocationHandler implements InvocationHandler {

2

public Object invoke(Object proxy, Method method, Object[] args) throws Throwable

3

return null;

4

}

5

}

надо бы поправить(:

Ответить

-

0

+

PaiMei in J#

Grand Master в Eagles' Claw

15 октября 2021, 11:49

...

Эти примеры кода взяты из разных частей статьи, если смотреть по логике написания, то никаких противоречий не возникает

Ответить

-

0

+

Begemoth

Software Architect в Сиблион

4 августа 2021, 12:35

...

Сами пишут, "проxy-объект можно создать только используя интерфейсы", а затем хотят получить абстракты класс Reader

1

Reader reader = (Reader)Proxy.newProxyInstance(new CustomInvocationHandler());

2

reader.close();

Прокси возвращает объект нового (динамического) имплементируемого интерфейсами класса, и раз мы хотим вызвать метод close() интерфейса Closeable,то надо к нему и приводить

1

Object proxy = Proxy.newProxyInstance(new CustomInvocationHandler());

2

Closeable reader = (Closeable)proxy;

3

reader.close();

или

1

Closeable reader = (Closeable)Proxy.newProxyInstance(new CustomInvocationHandler())

2

reader.close();

Ответить

-

+2

+

👍

Anonymous #2489173

Уровень 35

25 марта 2021, 08:17

...

с помощью которого фактически можно сконструировать объект во время исполнения программы (динамически), не создавая для него отдельного класса.

и после этого создаём класс и реализуем у него метод. шта?
да и зачем для создания объекта Ридера создавать класс? он ведь уже есть в стандартной библиотеке?
как всегда всё очень понятно

что произошло в последнем примере вообще непонятно. что-то передали и.... зачем?

НАЧАТЬ ОБУЧЕНИЕ

создали класс чтобы видеть с каким классом конкретно классом работаем.

Ответить

0

Артём Уколов

Уровень 38, Москва, Россия

12 марта 2021, 18:33

https://www.youtube.com/watch?v=aMc1wHveqGM тут понятнее

Ответить

+2

Vladimir “Rain_Senpai1995” Soldatenko

Уровень 35, Киев, Украина

17 января 2021, 22:32

Я так понял, что это еще один из способов обойти запрет множественного наследования через динамическое проксирование объекта

Ответить

0

Artem K.

Уровень 30, Москва

8 ноября 2020, 20:11

Вот в этой лекции все получше объяснялось: [От профессора](#)

Ответить

+18

Ильшат

Уровень 19, Сибай, Россия

8 октября 2020, 11:45

Бааааатюшки!

Ответить

+3

Показать еще комментарии

ОБУЧЕНИЕ

- Курсы программирования
- Курс Java
- Помощь по задачам
- Подписки
- Задачи-игры

СООБЩЕСТВО

- Пользователи
- Статьи
- Форум
- Чат
- Истории успеха
- Активности

КОМПАНИЯ

- О нас
- Контакты
- Отзывы
- FAQ
- Поддержка



JavaRush — это интерактивный онлайн-курс по изучению Java-программирования с нуля. Он содержит 1200 практических задач с проверкой решения в один клик, необходимый минимум теории по основам Java и мотивирующие фишки, которые помогут пройти курс до конца: игры, опросы, интересные проекты и статьи об эффективном обучении и карьере Java-девелопера.

ПОДПИСЫВАЙТЕСЬ

ЯЗЫК ИНТЕРФЕЙСА

Русский

СКАЧИВАЙТЕ НАШИ ПРИЛОЖЕНИЯ

Доступно в

Google Play

Загрузите в

App Store

