|  |  |
| --- | --- |
| **Java передает переменные только по значению? Проклятье...** |  |

|  |
| --- |
|  |
| Здесь приведен перевод статьи Scott Stanchfield "Java is Pass-by-Value, Dammit!" [1], раскрывающей смысл передачи параметров в методы на Java.  Термины семантики "pass-by-value" и семантики "pass-by-reference" относятся к параметрам функции, и имеют весьма точные определения. Но они иногда почему-то сильно искажаются, когда люди говорят о Java. Мне хотелось бы исправить это...  **Pass-by-value** (передача параметра по значению). Действительный параметр (или выражение в аргументе) полностью вычисляется, и значение результата копируется в отдельную ячейку памяти, предназначенную для хранения значения этого параметра во время выполнения функции. То есть функция имеет дело с копией переменной, которую передали в функцию в качестве параметра. Место в памяти под параметр - обычно кусок runtime-стека в приложении (обрабатываемый Java), но другие языки могут выбрать хранение параметра в другом месте.  **Pass-by-reference** (передача параметра по ссылке). Формальный параметр действует просто как псевдоним (alias) реального параметра. Функция или метод, которая использует формальный параметр (для чтения или записи), в действительности использует актуальный параметр, существующий где-то вне функции.  В Java жестко задано использовать вариант передачи параметров по значению (pass-by-value), точно так же, как это делается на языке C (однако на C можно также передавать параметр и по-другому, pass-by-reference). Подробности прочитайте в Java Language Specification (JLS), там все корректно разъяснено. В разделе "JLS 8.4.1. Formal Parameters" написано:  "*Когда запускается метод или конструктор, значение актуального аргумента инициализируют новые создаваемые переменные параметра, каждая с объявленным типом. Это все делается до запуска кода из тела метода или конструктора. Идентификатор, который появляется в DeclaratorId, может использоваться как простое имя в теле метода или конструктора для ссылки на формальный параметр.*"  Если коротко, то это можно выразить так: в Java есть указатели, и передача переменных происходит только по значению (pass-by-value). Никаких скрытых правил, все просто, тупо и однозначно (все также ясно и четко, как в дьявольском синтаксисе C++ ;).  Примечание: в конце этой статьи см. замечание по поводу семантики вызова remote-метода (RMI, remote method invocation). То, что обычно называют "pass by reference" (передача параметра по ссылке) для remote-объектов представлено чрезвычайно плохо организованной семантикой.  [**Лакмусовый тест**]  Здесь приведен простой тест для языка, поддерживающего семантику передачи параметра по ссылке (pass-by-reference): можете Вы написать обычную функцию/метод swap(a,b) на этом языке?  Традиционный метод или функция swap принимает два аргумента и переставляет их значения прямо в тех переменных, которые определены вне функции, и переданы в неё через параметры по ссылке. Базовая структура такой функции выглядит примерно так:  *//Как работает базовый код функции swap (не Java).*  swap(Type arg1, Type arg2)  {  Type temp = arg1;  arg1 = arg2;  arg2 = temp;  }  Если Вы можете на тестируемом языке написать такой метод или функцию, и вызвать его примерно так:  *//Вызов функции swap (не Java).*  Type var1 = ...;  Type var2 = ...;  swap(var1,var2);  И после этого вызова значения в переменных var1 и var2 поменяются местами, то тогда язык поддерживает семантику передачи параметров по ссылке (pass-by-reference). Например, на Pascal можно написать:  *//Функция swap на Pascal.*  **procedure** swap(**var** arg1, arg2: SomeType);  **var**  temp : SomeType;  **begin**  temp := arg1;  arg1 := arg2;  arg2 := temp;  **end**;  ...  **var**  var1, var2 : SomeType;  **begin**  var1 := ...; *{ value "A" }*  var2 := ...; *{ value "B" }*  swap(var1, var2);  *// теперь var1 имеет значение "B", и var2 имеет значение "A".*  **end**;  Или можно то же самое проделать на С++:  *//Функция swap на C++.*  void swap(SomeType& arg1, Sometype& arg2)  {  SomeType temp = arg1;  arg1 = arg2;  arg2 = temp;  }  ...  SomeType var1 = ...; *// value "A"*  SomeType var2 = ...; *// value "B"*  swap(var1, var2);  *// теперь var1 имеет значение "B", и var2 имеет значение "A".*  Но на Java этого сделать нельзя!  [**Теперь подробнее...**]  Здесь мы столкнулись со следующей проблемой: на Java объекты передаются по ссылке, а примитивные типы передаются по значению. Причем это корректно только наполовину. Любой может легко согласиться, что примитивы передаются по значению; в Java вообще нет такого понятия, как указатель/ссылка на примитив.  Однако объекты не передаются (явно) по ссылке. Корректным оператором была бы ссылка на объект, которая передается по значению в функцию/метод. В последующих примерах Java будет показано, в чем тут дело.  *//Java: пример передачи параметра по значению.*  **public** void foo(Dog d)  {  d = **new** Dog("Fifi"); *// создается собака "Fifi".*  }    Dog aDog = **new** Dog("Max"); *// создается собака "Max".*  *// В этом месте aDog указывает на собаку "Max".*  foo(aDog);  *// aDog все еще указывает на собаку "Max".*  Переданная переменная (aDog) не была модифицирована! После вызова foo, aDog все также указывает на собаку "Max"!  Многие люди ошибочно думают, что получится состояние наподобие следующего:  **public** void foo(Dog d)  {  d.setName("Fifi");  }  Здесь показано, что Java фактически передает объекты по ссылке. Ошибочно трактуют определение:  Dog d;  Когда Вы задаете такое определение, то определяете указатель на объект Dog, не сам объект Dog.  [**Указатели в сравнении с ссылками**]  Проблема здесь в том, что разработчики Sun допустили ошибку в назначении понятий. В терминологии разработки языка программирования термин "pointer" (указатель) это переменная, которая неявно отслеживает место размещения некой порции данных. Значение указателя часто адрес интересующих нас данных. Некоторые языки позволяют манипулировать этим адресом, другие не позволяют этого делать.  Термин "reference" (ссылка) означает псевдоним (alias) другой переменной. Любая манипуляция с reference-переменной напрямую изменяет оригинальную переменную. Прочитайте следующее высказывание в документации "JLS 4.3. Reference Types and Values":  "*Переменные-ссылки (reference values, часто просто references) являются указателями на эти объекты, и есть специальная ссылка null, которая говорит, что не указан никакой объект.*"  Они подчеркивают "указатели" (pointers) в своем описании... Интересненько...  Когда первоначально создавался Java, у них был все-таки "указатель" (Вы можете в реальности видеть атавизмы от этого в виде появления ошибок типа NullPointerException). Sun хотел продвигать Java в качестве безопасного языка, и одним из достоинств Java преподносился запрет на использовании арифметики указателей, которая имеется в C++.  Разработчики пошли по пути использования другого имени для концепции указателей, формально называя их "ссылками" (references). Мне кажется это большой ошибкой, которая создает еще больше путаницы в процессе программирования.  Хорошее объяснения понятия ссылочных переменных дано в статье [2] (оно дано в специфике C++, однако здесь сказаны правильные вещи про всю концепцию reference-переменной). Слово "reference" в разработке языка программирования изначально произошло от способа, каким Вы передаете данные подпрограммам/функциям/процедурам/методам. Ссылочный параметр (reference parameter) является псевдонимом (alias) переменной, переданной в качестве параметра.  В результате Sun сделала ошибку и подменила понятия, что вызвало беспорядок. Все-таки в Java есть указатели (несмотря на то, что арифметика указателей запрещена), и если Вы признаете это, то в поведении Java найдете намного больше смысла.  [**Вызовы методов**]  *//Тут происходит передача указателя по значению:*  foo(d);  Вызов метода foo в этом примере передает значение d; здесь вовсе не передается объект, на который указывает d!  Значение передаваемого указателя аналогично адресу памяти. Под капотом в тонкостях могут быть отличия, но Вы можете думать об этом именно таким образом. Переданное значение уникально идентифицирует некий объект, находящийся в куче (heap).  Однако не имеет никакого значения, как указатели реализованы внутри Java. Вы программируете с ними на Java точно так же, как это делали бы на C или C++. Синтаксис незначительно поменялся: еще один плохой выбор в дизайне Java; здесь надо было для разыменования (de-referencing) использовать тот же синтаксис, как в C++. Например, на Java объявление:  *//Java: пример объявления указателя.*  Dog d;  означает абсолютно то же самое, что и на C++ объявление:  *//C++: пример объявления указателя.*  Dog \*d;  Используются указатели при этом тоже одинаково:  *//Java: вызов метода по указателю.*  d.setName("Fifi");  *//C++: вызов метода по указателю.*  d->setName("Fifi");  Итак, можно сделать следующий вывод: Java имеет указатели, и в случае использования объектов в виде параметров функции в функцию передается значение указателя. Нет никакого способа передать сам объект в качестве параметра. Вы можете передать только указатель на объект. Имейте в виду, что когда делается вызов:  *//Java: тут на самом деле передается значение указателя.*  foo(d);  то Вы не передаете объект. Вы передаете указатель на объект. Немного другой взгляд (но все-таки корректный) на эту проблему Вы также можете найти в отличной книге Peter Haggar "Practical Java".  [**Remote Method Invocation (RMI)**]  Когда передаются параметры в remote-методы, все становится несколько сложнее. Во-первых, мы (обычно) имеем дело с передачей данных между двумя независимыми виртуальными машинами, которые могут находиться также на разных физических машинах. Передача значения указателя не принесла бы пользы, потому что целевая виртуальная машина не имеет доступа к куче виртуальной машины, вызвавшей метод.  Вы будете часто встречаться с понятиями "pass by value" и "pass by reference", используемыми в контексте RMI. Причем они имеют некое "логическое" значение, и в реальности некорректны для надлежащего использования.  Вот что обычно подразумевается под этими фразами в контексте RMI. Обратите внимание, что здесь теряется традиционное значение терминов семантики "pass by value" и "pass by reference":  **RMI Pass-by-value**: действительный параметр сериализируется и передается по сетевому протоколу на целевой отдаленный объект. Сериализация по-существу "сжимает" данные от исходных данных объекта/примитива. На принимающем конце данные используются для построения "клона" оригинального объекта или примитива. Имейте в виду, что этот процесс может быть довольно дорогим по ресурсам, если фактические параметры указывают на большие объекты (или большие графы объектов). Это не вполне правильное использования термина "pass-by-value"; возможно, это надо было бы назвать "pass-by-memento".  **RMI Pass-by-reference**: действительный параметр, который сам по себе remote-объект, предоставлен через прокси. Этот прокси отслеживает место, где живет фактический параметр, и в любое время, когда целевой метод использует формальный параметр, происходит вызов другого remote-метода, чтобы передать обратно действительный параметр. Это может быть полезным, если актуальный параметр указывает на большой объект (или граф объектов), и здесь используется несколько обратных вызовов. Здесь использование термина "pass-by-reference" также не совсем корректно (Вы не можете поменять действительный параметр сам по себе). Лучше бы подошло название типа "pass-by-proxy".  [**Обсуждение на stackoverflow.com**]  Спецификация Java утверждает, что всегда передача параметров происходит по принципу pass-by-value. В Java нет такого понятия, как "pass-by-reference". Ключевым понятием для понимания является следующее - здесь на Java переменная myDog не является собакой, это указатель на собаку:  Dog myDog;  То есть, если Вы написали следующее:  Dog myDog = **new** Dog("Rover");  foo(myDog);  то Вы на самом деле передали в функцию foo адрес созданного объекта Dog (возможно это не физический адрес как таковой, но это самый простой способ правильно понимать, что происходит).  Предположим, что объект Dog находится в памяти по адресу 42. Это означает, что методу foo будет передано 42. Пусть метод foo определен так:  **public** void foo(Dog someDog)  {  someDog.setName("Max"); *// AAA*  someDog = **new** Dog("Fifi"); *// BBB*  someDog.setName("Rowlf"); *// CCC*  }  Посмотрите, что произойдет. Параметр someDog установлен в значение 42.  На строке "AAA": someDog следует за объектом Dog по указателю (который находится по адресу 42), так что для этого Dog (находящегося по адресу 42) будет запрошено изменение имени на Max.  На строке "BBB": будет создан новый объект Dog. Новый объект получит новый адрес в памяти. Предположим, что этот адрес 74, так что указателю someDog будет присвоено значение 74.  На строке "CCC": someDog следует за объектом Dog по указателю (который находится по адресу 74), так что для этого Dog (находящегося по адресу 74) будет запрошено изменение имени на Rowlf. После этого выполнение метода foo заканчивается.  Давайте теперь подумаем, что же произойдет вне метода foo: изменится ли myDog?  Это ключевой момент. Если иметь в виду, что myDog является указателем, а не просто объектом Dog, то ответ будет НЕТ. myDog все еще имеет значение 42, как это и было до вызова функции; он все еще указывает на оригинальный объект Dog. Совершенно допустимо следовать по адресу myDog для изменения содержимого объекта, адрес myDog при этом остается однако неизменным.  Java работает абсолютно так же, как это делает C. Вы можете назначить указатель, передать указатель методу, следовать по указателю и изменять данные, на которые указатель указывает. Однако Вы не можете поменять место расположения объекта, на который этот указатель указывает.  На C++, Ada, Pascal и других языках, которые поддерживают pass-by-reference, Вы можете в действительности поменять переменную, которая была передана.  Если бы у Java была семантика pass-by-reference, то вышеопределенный нами метод изменился бы, и myDog указывал бы на другой объект, который был присвоен someDog на строке BBB.  Думайте о reference-параметрах как о псевдонимах для переданной переменной. Когда присвоено значение псевдониму, то это была переданная в метод переменная. |