



Урок 8

Подготовка к собеседованию Java Middle

Обзор вопросов с собеседований на должность Java Middle

[Что должен знать Java Middle Developer](#)

[Задачи с собеседований](#)

[Что такое SOAP и какой формат он поддерживает](#)

[REST и правила архитектуры](#)

[JSON](#)

[Deadlock](#)

[Реализуйте класс equals\(\)](#)

[Многопоточность. Плюсы и минусы.](#)

[java.io.BufferedReader vs java.util.Scanner](#)

[hashCode\(\)](#)

[Логические задачи с собеседований](#)

[Продолжите последовательность](#)

[Как вычислить 2 в 64 степени. не пользуясь калькулятором](#)

[Новые возможности Java 9](#)

[JShell \(REPL\)](#)

[Фабричные методы для создания иммутабельных коллекций](#)

[Улучшение блока try-with-resources](#)

[Улучшение аннотации @Deprecated](#)

[Изменения в интерфейсах](#)

[HTTP/2 Client](#)

[Process API](#)

[Дополнительные материалы](#)

[Используемая литература](#)

Что должен знать Java Middle Developer

Набор знаний Java-Middle-девелопера значительно расширяется после стадии Junior. На плечи разработчика ложится ряд вопросов, касательно знания не только синтаксиса, но и сторонних библиотек, знание аннотаций и многого другого. Проведя небольшой анализ, можно сделать вывод о том, какие основные моменты должен знать разработчик этого уровня:

- Serialization;
- Maven;
- IO;
- Паттерны программирования;
- TDD;
- всё то, что было изучено с позиции Java Junior;
- SOAP;
- REST;
- JSON;
- XML.

С сериализацией мы познакомились ранее, когда писали тестовый проект. Для передачи файлов мы использовали стандартный класс File, который, кстати, реализует интерфейс сериализации.

С maven мы познакомились в третьем уроке текущего курса, разобрались с основными возможностями и научились тянуть проекты из удалённых репозиториях к себе.

С вводом/выводом мы работаем уже не первый курс, так как даже команда вывода на консоль, это тоже операция ввода-вывода. Тем не менее, описав архитектуру с передачей файлов по сети, мы научились отправлять и получать данные с потоков, взятых из сокетов, правильно их обрабатывать и отправлять ответные сообщения.

Паттернов программирования великое множество, при создании архитектуры мы рассмотрели основные из них. Конечно, не нужно намеренно искать места для того, чтобы внедрить тот или иной паттерн, однако стоит понимать заранее, если вы хотите получить более гибкий интерфейс, возможно, то, что вам нужно, уже давно написано.

Тестирование кода - неотъемлемая часть проектирования, поэтому тестирование также входит в обязанности Java-Middle-девелопера.

Задачи с собеседований

Что такое SOAP и какой формат он поддерживает

Вопрос касательно протоколов обмена данными очень остро стоит на собеседованиях и в работе Java-программиста остаётся надолго.

SOAP (Simple Object Access Protocol) — простой протокол доступа к объектам. Если смотреть в историю этого протокола, то изначально он предназначался для удалённого вызова процедур, сегодня это один из самых популярных протоколов обмена сообщениями. SOAP общается не простыми текстовыми полями, а привычной нам разметкой XML. Пример простого SOAP-запроса на сервер представлен ниже:

```
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
  <soap:Body>
    <getProductDetails xmlns="http://warehouse.example.com/ws">
      <productID>12345</productID>
    </getProductDetails>
  </soap:Body>
</soap:Envelope>
```

Ответ же на этот запрос представлен ниже:

```
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
  <soap:Body>
    <getProductDetailsResponse xmlns="http://warehouse.example.com/ws">
      <getProductDetailsResult>
        <productID>12345</productID>
        <productName>Стакан граненый</productName>
        <description>Стакан граненый. 250 мл.</description>
        <price>9.95</price>
        <currency>
          <code>840</code>
          <alpha3>USD</alpha3>
          <sign>$</sign>
          <name>US dollar</name>
          <accuracy>2</accuracy>
        </currency>
        <inStock>true</inStock>
      </getProductDetailsResult>
    </getProductDetailsResponse>
  </soap:Body>
</soap:Envelope>
```

Как правило, SOAP используется как надстройка над HTTP.

REST и правила архитектуры

REST (REpresentation State Transfer) или передатчик состояния представления является архитектурным стилем взаимодействия компонентов распределённой системы (к примеру, клиентов и серверов). Некоторые считают, что REST является упрощённой альтернативой SOAP, но на самом деле REST - это просто стиль построения архитектуры, а SOAP - это целый протокол.

По сути, REST может использовать много протоколов: HTTP для передачи данных, JSON (рассмотрим далее), XML для упаковки данных и URL. Как уже было сказано, REST - всего лишь стиль, и, как любой другой, имеет свои характерные отличия. Ниже представлены ограничения, в рамках которых реализуется REST:

1. Модель клиент-сервер. Разграничение этих двух понятий очень важно и означает, что клиент - это нечто отдельное от сервера, и сервер также отделён от клиента. Понять это сложно, однако я попробую выразиться немного иначе: давайте представим, как будто сервер ничего не знает о клиенте. Значит, задача клиента - дать серверу для ответа как можно больше информации. Это похоже на магазин: вы приходите за хлебом, вы знаете его стоимость, где он лежит и у вас есть с собой деньги, вы знаете, как пройти на кассу и как оплатить покупку. В этом примере вы - клиент, а магазин - это сервер, так как магазин понятия не имеет, что Вы собираетесь сейчас прийти и купить хлеб.
2. Отсутствие состояния. Вытекает из предыдущего примера. Для того чтобы купить хлеб, вам не нужно сообщать продавцу о наличии у Вас каких-то хронических болезней, квартиры и авто в личном пользовании, магазин может только выдать вам хлеб и получить от вас деньги, никакая другая информация ему не нужна.
3. Кэширование. Клиент сохраняет некую "временную" информацию для сохранения ресурсов для запросов. Опять же идём в магазин и покупаем уже не хлеб, а газету. Формально мы пришли и купили информацию и храним её у себя, информация в газете устаревает через неделю и мы идём и покупаем новую. Мы не ходим в магазин каждые 5 минут, для того, чтобы проверить, есть ли в газете в магазине какая-то новая информация. Соответственно, мы храним у себя информацию для последующего использования. Значит кэш у нас, как у клиентов равен одной неделе.
4. Единообразие интерфейса. Означает, что клиент и сервер общаются на "одном языке". Опять аналогия с магазином: Мы идём в магазин для покупки хлеба, вы и сотрудник магазина говорите на одном языке, и сотрудник готов принять от вас оплату именно в той валюте, которая имеется у вас, соответственно, у Вас и у магазина имеется один и тот же интерфейс.
5. Слои. Наличие слоев помогает разгрузить нагрузку. Аналогия с магазином до сих пор в обиходе. Все магазины первоначально закупают свои продукты у поставщика. Представьте, если бы магазинов не было, и все ходили за продуктами прямо к поставщику, какие бы очереди были на кассах? Абсолютно то же самое происходит и в этом случае, наличие промежуточных узлов упрощает работу с сервисом.
6. Код по требованию. Означает, что сервер может расширить функциональность клиента путём отправки скриптов или другого кода. И здесь можно привести некую аналогию с магазином. Вы идёте и покупаете себе сотовый телефон. Получается, что магазин расширил Вашу функциональность, и Вы можете общаться не только с помощью бумажных писем, но и с помощью телефона.

Если обратить внимание на некоторые популярные сервисы, к примеру, на социальную сеть vk.com, то даже она предоставляет свой REST API для работы в Ваших приложениях, который полностью удовлетворяет требованиям, описанным выше.

JSON

JSON (JavaScript Object Notation) - это не протокол и не стиль, это текстовый формат для обмена сообщениями. В настоящее время набирает большую популярность, выводя XML на второй план. Несмотря на JavaScript в названии, этот формат развивается отдельно от этого языка. Преимущественно над XML обладает очень хорошей сериализацией сложных структур в текстовом формате и представляет все данные в формате ключ:значение:

```
{
  "firstName": "Иван",
  "lastName": "Иванов",
  "address": {
    "streetAddress": "Ереванская ул. 28б кв. 8",
    "city": "Москва",
    "postalCode": 101101
  },
  "phoneNumbers": [
    "812 123-1234",
    "916 123-4567"
  ]
}
```

Deadlock

Вопрос касательно синхронизации также часто стоит на собеседованиях. Что такое Deadlock, почему он возникает, и какие есть способы избежать его?

Ответ: Deadlock (взаимная блокировка) - состояние, при котором несколько потоков находятся в режиме ожидания ресурсов, занятых этими потоками. Работа потоков приостанавливается и программа "зависает".

Пример: в объекте А имеется ссылка на объект Б, который имеет ссылку на объект А. При выполнении первый поток вызывает второй поток, встаёт на блокировку, а второй поток, вызывая первый, также встаёт в блокировку, в результате чего возникает DeadLock. Как бороться с этим недугом в программировании? Синхронизировать все объекты в одном и том же порядке. Оба потока могут зависеть и проверять также третий независимый объект.

В нашем примере можно переписать объект А и Б на третий второстепенный объект, в котором и находится механизм разрешения ситуаций взаимной блокировки, после чего просто синхронизировать методы обоих объектов.

Реализуйте класс equals()

Самым распространённым практическим заданием является реализация таких методов , как equals() и compare(). На собеседованиях задача звучит так:

Дан класс Point, предложите реализацию метода equals, чтобы обеспечить корректность сравнения , в том числе и математического.

```

public class Point {
    private double x;
    private double y;
    public Point(double x, double y) {
        this.x = x;
        this.y = y;
    }
    @Override
    public boolean equals(Object obj) {
    }
    @Override
    public int hashCode() {
    }
}

```

Ответ на данную задачу можно реализовать следующим образом:

```

@Override
public boolean equals(Object other) {
    if(!super.equals(other)) return false;
    if (this == other) return true;
    if (other == null) return false;
    if(this.getClass() != other.getClass()) return false;
    Point otherObj = (Point)other;
}

```

Если имеются поля других типов, то сравнение через == не подойдёт, так как в задаче также написано, что сравнение должно быть также и математическим. Соответственно, значения мы будем сравнивать методом equals() каждого поля.

Многопоточность. Плюсы и минусы.

Что можно сказать о многопоточности в программировании? Оцените плюсы и минусы?

Как и в любом деле, в использовании многопоточности есть свои достоинства и недостатки. Важно, чтобы в каждом конкретном случае недостатки не перевешивали достоинств.

Достоинства:

- Несколько потоков одновременно обрабатывают данные разного характера, при этом при остановке одного из потоков действие программы не завершается, и остальные потоки работают независимо от остановленного. Конечно, это работает только в том случае, если между потоками нету никаких взаимодействий, и они действительно являются независимыми. Но данная проблема решается довольно тривиально: java поддерживает всевозможные методы для синхронизации данных между потоками.

Недостатки:

- Многопоточность требует отдельных усилий и контроля от разработчика.

- Высокие требования к аппаратной части, так как отдельный поток работает в режиме отдельного процесса.

Выводы:

Все выводы можно делать только исходя из первородной задачи. На самом деле, в случаях, когда можно обойтись без использования многопоточности, лучше не создавать себе лишнюю работу и обходиться без неё, однако в современном мире уже практически нет задач, в которых отсутствие многопоточности можно назвать хорошим качеством.

java.io.BufferedReader vs java.util.Scanner

Есть ли разница между java.io.BufferedReader и java.util.Scanner?

Ответ: BufferedReader предназначен для чтения потока символов с буферизацией.

Ответ: BufferedReader предназначен для чтения буферизированного потока символов тогда, когда Scanner подходит только для разбора входного набора данных на составляющие (автоматически форматирует, вставляет разделители).

Их единственное сходство - возможность чтения входящего набора данных (файла, к примеру), и только! Причём скорость чтения файла посредством класса java.io.BufferedReader выше, нежели посредством использования класса java.util.Scanner

hashCode()

Для чего нужен метод hashCode()?

Ответ:

Для вычисления хэш-функции объекта. На самом деле хэш объекта - это и есть его идентификатор. Собственно оператор "==" проверяет на сходство хэш-функции каждого объекта, т.е. в этот набор данных входит вся информация об объекте, включая адрес расположения.

Например, рассмотрим вставку элементов в HashMap:

```
public V put(K key, V value) {  
    ...  
    int hash = hash(key.hashCode());  
    int i = indexFor(hash, table.length);  
    ...  
    addEntry(hash, key, value, i);  
    ...  
}
```

Как мы видим, i-е место вставки объекта вычисляется при помощи хэша. А для вычисления нам нужна хорошая хэш-функция, чтобы давала равномерное распределение и поменьше коллизий.

То есть ответ на вопрос заключается в том, что существуют коллекции (HashMap, HashSet), которые используют хэш-код, как основу при работе с объектами. А если хэш для равных объектов будет разным, то в HashMap будут два равных значения, что является ошибкой. Поэтому необходимо соответствующим образом переопределить метод hashCode().

Логические задачи с собеседований

Продолжите последовательность

Собеседования в Google проходят иногда с очень интересными задачами. На сегодня вопрос стоит следующим образом: продолжите последовательность:

10, 9, 60, 90, 70, 66

Ответ очень неоднозначен, но выглядит интересно:

Что в первую очередь хочется обнаружить в этом беспорядочном распределении чисел? Конечно же закономерность, однако, я Вас уверяю, здесь вы её не найдёте. На самом деле эта задача не из технической должности, однако её решение хорошо расширяет абстракцию при программировании. Ключевой аспект в данной задаче - здесь нет математики. Но стоит произнести эти числа английским, как возникает небольшая подсказка, дающая нам выход во второй раунд. Буквы расположены в порядке возрастания.

ten
nine
sixty
ninety
seventy
sixty-six

После детального рассмотрения можно понять, что 10 в данной последовательности почему-то не единственное число из трёх букв. Но на этом месте не 2, и не 1, а именно 10 и такая же ситуация с числом 9. Значит в списке включены самые большие числа из тех, которые можно написать за такое же количество символов.

Так какой будет правильный ответ? Очевидно, что в числе, следующем за 66, должно быть девять букв (не считая возможного дефиса), и оно должно быть самым крупным в своём роде. Немного подумав, можно сказать, что ответ будет 96 (ninety-six). Вы понимаете, что сюда не подходят числа, превышающие 100, поскольку для «one hundred» уже нужно десять букв.

Может быть, у вас возникнет вопрос, почему в приведённом списке на месте 70 не стоит сто (hundred), или миллион, или миллиард, для написания которых также нужно семь букв. Скорее всего потому, что на правильном английском языке говорится не «сто», а «одна сотня», то же относится и к двум другим случаям.

Казалось бы, всё, вот он правильный ответ. В Google его считают приемлемым, но не самым совершенным. Есть число побольше:

[illegible]

Однако и это еще не самый лучший вариант. Идеальный ответ: «ten googol», десять гуголов.

Как вычислить 2 в 64 степени, не пользуясь калькулятором

Приведём один из вариантов возможных рассуждений. Любой инженер знает, что $2^{10} = 1024$. Будем считать, что это приблизительно 1000. Умножим 2^{10} на себя шесть раз и получим 2^{60} . Это около 1000 в шестой степени или 10^{18} , также известное как квинтиллион. Осталось только умножить его на 24 (16), чтобы получить искомое 2^{64} . Таким образом, очень приблизительный, но быстрый ответ будет 16 квинтиллионов.

На самом деле, чуть больше, т.к. 1024 на 2.4% больше 1000. Мы используем это приближение 6 раз, и поэтому ответ должен быть чуть более, чем на 12% больше. Это добавляет еще 2 квинтиллиона. Поэтому более точно будет 18 квинтиллионов.

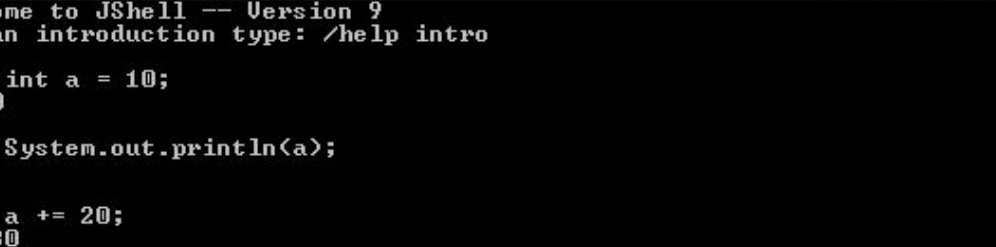
Точное значение: 18 446 744 073 709 551 616

Есть ещё один быстрый хак. Многие знают, что максимальное число 32-битного unsigned int — это что-то около 4 миллиардов т.е. $2^{32} \approx 4 \times 10^9$. Осталось только умножить это само на себя и получить около 16—17 квинтиллионов.

Новые возможности Java 9

JShell (REPL)

В Java 9 представлен новый инструмент под названием **JShell**, также известный как REPL (Read Evaluate Print Loop), предназначенный для быстрой и легкой проверки кодовых конструкций, таких как классы, интерфейсы, перечисления, объекты, выражений.



```
C:\Program Files\Java\jdk-9\bin\jshell.exe

! Welcome to JShell -- Version 9
! For an introduction type: /help intro

jshell> int a = 10;
a ==> 10

jshell> System.out.println(a);
10

jshell> a += 20;
33 ==> 30

jshell> int b = 5;
b ==> 5

jshell> int c = a + b;
c ==> 35

jshell> System.out.println("c = " + c);
c = 35

jshell>
```

Ниже представлен пример написания кода в JShell.

```
G:\>jshell
| Welcome to JShell -- Version 9-ea
| For an introduction type: /help intro
jshell> int a = 10
a ==> 10
jshell> System.out.println("a value = " + a )
a value = 10
```

Фабричные методы для создания иммутабельных коллекций

В Java 8 иммутабельные коллекции создаются через статические методы класса Collections: Collections.unmodifiableList(), Collections.unmodifiableSet(), Collections.unmodifiableMap().

```
public class Main {
    public static void main(String[] args) {
        List<String> unmodList = Collections.unmodifiableList(Arrays.asList("A", "B", "C"));
        Set<String> unmodSet = Collections.unmodifiableSet(new HashSet<>(Arrays.asList("A", "B",
"C"))));
    }
}
```

Выглядят такие записи довольно громоздко и для улучшения читаемости кода в интерфейсы List, Set и Map был добавлен метод of(), который и занимается созданием иммутабельной коллекции. В интерфейсе Map дополнительно добавлен метод ofEntries(), который позволяет добавить в отображение иммутабельную пару ключ-значение. Пример работы с новыми методами представлен ниже.

```
public class Main {
    public static void main(String[] args) {
        List<String> immutableList1 = List.of();
        List<String> immutableList2 = List.of("one", "two", "three");
        Map<Integer, String> emptyImmutableMap1 = Map.of();
        Map<Integer, String> nonemptyImmutableMap2 = Map.of(1, "one", 2, "two", 3, "three");
    }
}
```

Улучшение блока try-with-resources

В Java 9 произошли изменения с блоками try-with-resources. Если мы имеем объявление некоего ресурса за пределами блока, и этот ресурс объявлен как final или effectively final, то этот ресурс

можно подавать в блок try-with-resources без дополнительных переменных, которые были нужны в Java 7 или 8.

```
public class Main {
    void doSomething() throws IOException {
        BufferedReader br = new BufferedReader(new FileReader("data.txt"));
        try (br) {
            System.out.println(br.readLine());
        }
    }
}
```

Улучшение аннотации @Deprecated

В Java 8 и более ранних версиях аннотация @Deprecated была всего лишь маркерным интерфейсом, который указывал что класс, поле, метод, интерфейс, конструктор и т.д. устарели. В Java 9 в эту аннотацию были добавлены два метода forRemoval() и since(), а также инструменты анализа устаревшего кода.

```
@Deprecated(since = "0.4.1")
public void veryOldMethod() {
    System.out.println("Java");
}
```

Изменения в интерфейсах

В Java 9 появилась возможность создавать методы с модификатором доступа **private**, которые обязательно должны иметь тело (т.е. методы не должны быть абстрактными). Такие методы могут быть использованы только внутри интерфейса, в котором объявлены.

```
public interface MyInterface {
    private void privateMethod() {
        System.out.println("Java");
    }
}
```

HTTP/2 Client

Добавлен новый HTTP/2 Client API, который поддерживает работу с протоколом HTTP/2 и WebSocket. Существующий HTTP Client (URLConnection API) имеет несколько недостатков: поддерживает только HTTP/1.1 и не поддерживает HTTP/2 и WebSocket, работает только в блокирующем режиме что может приводить к снижению производительности).

Новая версия клиента находится в пакете `java.net.http`, поддерживает протоколы HTTP/1.1 и HTTP/2, может работать как в синхронном (блокирующем) режиме, так и в асинхронном, поддерживает работу с WebSocket в асинхронном режиме.

Пример запроса на авторизацию через GET-запрос:

```
public class Main {
    public static void main(String[] args) throws Exception {
        HttpClient client = HttpClient.newBuilder()
            .authenticator(new Authenticator() {
                @Override
                protected PasswordAuthentication getPasswordAuthentication() {
                    return new PasswordAuthentication("username", "password".toCharArray());
                }
            })
            .build();

        HttpRequest request = HttpRequest.newBuilder()
            .uri(new URI("https://www.geekbrains.ru"))
            .GET()
            .build();

        HttpResponse<String> response = client.send(request, HttpResponse.BodyHandler.asString());
        System.out.println(response.statusCode());
        System.out.println(response.body());
    }
}
```

Пример асинхронного запроса:

```

public class Main {
    public static void main(String[] args) throws Exception {
        HttpClient client = HttpClient.newHttpClient();

        HttpRequest request = HttpRequest.newBuilder()
            .uri(new URI("https://www.geekbrains.ru/"))
            .GET()
            .build();

        CompletableFuture<HttpResponse<String>> response = client.sendAsync(request,
            HttpResponse.BodyHandler.asString());

        Thread.sleep(5000);
        if(response.isDone()) {
            System.out.println(response.get().statusCode());
            System.out.println(response.get().body());
        } else {
            response.cancel(true);
            System.out.println("Запрос выполняется более 5 секунд... отмена.");
        }
    }
}

```

Process API

До Java 9 в языке имелись довольно ограниченные возможности по управлению процессами операционной системы. Например, для того чтобы просто получить PID (**P**rocess **I**dentifier, идентификатор процесса) процесса необходимо было использовать нативный код и какие-то обходные пути. Кроме того, это требовало различных реализаций для каждой платформы, чтобы гарантировать, что код будет выдавать нужный результат.

В Java 9 для получения PID процесса в ОС Linux код:

```

public class Main {
    public static void main(String[] args) throws Exception {
        Process proc = Runtime.getRuntime().exec(new String[]{"/bin/sh", "-c", "echo $PPID"});
        if (proc.waitFor() == 0) {
            InputStream in = proc.getInputStream();
            int available = in.available();
            byte[] outputBytes = new byte[available];
            in.read(outputBytes);
            String pid = new String(outputBytes);
            System.out.println("Your pid is " + pid);
        }
    }
}

```

Превратится в код:

```
public class Main {  
    public static void main(String[] args) throws Exception {  
        Process proc = Runtime.getRuntime().exec(new String[]{"/bin/sh", "-c", "echo $PPID"});  
        System.out.println("Your pid is " + proc.pid());  
    }  
}
```

Данное обновление позволяет Java лучше взаимодействовать с ОС: узнавать PID процессов, работать с их именами и состоянием, получать списки процессов и пр.

Дополнительные материалы

1. [Собеседование Java](#)
2. [канал на YouTube - 300+ вопросов с собеседований](#)
3. [10 интересных вопросов на tproger](#)

Используемая литература

Для подготовки данного методического пособия были использованы следующие ресурсы:

1. Брюс Эккель - Философия Java
2. [Stack Overflow \(Java Tag\)](#)
3. <http://www.journaldev.com/13121/java-9-features-with-examples>
4. <http://www.journaldev.com/12942/javase9-factory-methods-immutable-list>
5. <http://www.journaldev.com/12850/java9-private-methods-in-interface>
6. <https://dzone.com/articles/5-features-in-java-9-that-will-change-how-you-deve>
7. <https://labs.consol.de/development/2017/03/14/getting-started-with-java9-httpclient.html>