lichMax

40 уровень

Санкт-Петербург

23.07.2017 19770 6

Уровень 40. Ответы на вопросы к собеседованию по теме уровня

Собственно, такие вопросы были на этом уровне:

1. Что такое IP-адрес?
2. В чем отличие host и domain?
3. Какие методы в HTTP вы знаете
4. Чем отличаются методы GET, POST и HEAD?
5. Что такое REST?
6. Зачем нужен класс Calendar в Java?
7. Как преобразовать дату в Java к нужному формату?
8. В чем отличие URI и URL?
9. Что такое сокеты?
10. Отличие классов Socket и URL?

А вот мои ответы:

1. **IP-адрес** — это уникальный сетевой адрес узла в компьютерной сети, построенной на стеке протоколов TCP/IP. В сети Интернет требуется глобальная уникальность адреса; в случае работы в локальной сети требуется уникальность адреса в пределах сети. В версии протокола IPv4 IP-адрес имеет длину 4 байта, а в версии протокола IPv6 IP-адрес имеет длину 16 байт. Обычно IP-адрес в версии протокола IPv4 записывают в виде четырёх десятичных чисел со значениями от 0 до 255, разделённых точкой, например, 192.168.0.3.

2. **Домен** — это адрес сайта или определённая зона, которая имеет своё имя, не похожее ни на какое другое имя в системе доменных имён. Домены бывают первого уровня, второго уровня, третьего уровня и т.д. Обычно домен первого уровня не доступен обычным пользователям для регистрации (примеры доменов первого уровня - ".ru", ".com", ".net"). Обычно домены третьего и следующих уровней называют субдоменами.
   Хост — это определённый компьютер или сервер, подключенный к локальной или глобальной сети. Хост обладает обладает уникальным адресом в среде сервисов TCP/IP (IP-адресом).

3. GET, POST, PUT, DELETE, OPTIONS, HEAD, PATCH, TRACE, LINK, UNLINK, CONNECT.

4.

	GET	**POST**	**HEAD**
Тело Запроса	Нет	Есть	Нет
Тело Ответа	Да	Да	Нет
Кеширование результата Запроса	Да	Нет	Да, заголовки
Идемпотентность	Да	Нет	Да

   Метод **GET** используется для запроса содержимого указанного ресурса. Метод **POST** применяется для передачи пользовательских данных заданному ресурсу. Метод **HEAD** обычно применяется для извлечения метаданных, проверки наличия ресурса (валидация URL) и чтобы узнать, не изменился ли он с момента последнего обращения. Метод HEAD аналогичен методу GET, за исключением того, что в ответе сервера отсутствует тело. Метода GET считает упрощённой версией POST, потому как метод GET не предполагает полноценного запроса, только URL в качестве такового.

5. **REST** — это архитектурный стиль взаимодействия компонентов распределённого приложения в сети. Термин был введён Роем Филдингом в 2000 году. Также им были введены требования, которым должно удовлетворять распределённое приложение, чтобы соответствовать архитектуре REST (такие приложения ещё называют RESTful). Вот эти требования:

   1. Модель "Клиент-Сервер" (означает, что сеть должна состоять из клиента и сервера; сервер - это тот, кто обладает ресурсами, клиент - тот, который их запрашивает))
   2. Отсутствие состояния (означает, что ни клиент, ни сервер не отслеживают состояния друг друга)
   3. Кеширование (клиенты и промежуточные узлы могут кешировать результаты запросов; соответственно, ответы сервера должны иметь явное или неявное обозначение, что они кешируемые или некешируемые)
   4. Единообразие интерфейса (означает, что между клиентами и серверами существует общий язык взаимодействия, который позволяет им быть заменяемыми или изменяемыми, без нарушения целостности системы):
      - Определение ресурса (означает, что каждый ресурс должны быть обозначен постоянным идентефикатором)
      - Управление ресурсами через представление (означает, что клиент хранит ресурс в виде его представления, и при желании изменения ресурса он отправляет серверу информацию о том, в каком виде он хотел бы видеть этот ресурс; сервер же рассматривает этот как запрос как предложение, и сам решает, что делать ему с хранимым ресурсом)
      - Самодостаточные сообщения (каждое сообщение содержит достаточно информации, чтобы понять, как его обрабатывать)
      - Гипермедиа (означает, что клиенты изменяют состояние системы только через действия, которые динамически определены в гипермедиа на сервер)
      - Система слоёв (означает, что в системе может быть больше двух слоёв (клиент и сервер), и при этом каждый такой слой знает только о своих соседних слоях, и не знает об остальных слоях, и взаимодействует только с соседними слоями)
      - Код по требованию (означает, что функциональность клиента может быть расширения за счёт загрузки кода с сервера в виде апплетов или сценариев)

   Удовлетворение этим требованиям позволяет добиться следующего:

   - Надёжность
   - Производительность
   - Масштабируемость
   - Прозрачность взаимодействия
   - Простота интерфейсов
   - Портативность компонентов

- Лёгкость внесения изменений
- Способность эволюционировать, приспосабливаясь к новым требованиям

6. Он нужен для более удобной работы с датой и временем. Он позволяет работать с датой в рамках календаря, то есть позволяет прибавлять и отнимать дни от какой-то конкретной даты, причём будут учитывать и високосные года. Кроме того, он позволяет представить время миллисекундах в удобном виде - год, месяц, день, часы, минуты, секунды. Также есть много методов для установки и получения разных параметров даты и времени, например: день недели, день месяца, день в году, номер недели в месяце, номер недели в году.

7. Для этого существует удобный класс `SimpleDateFormat`. Экземпляру этого класс можно передать шаблон представления даты, и тогда он в таком виде будет возвращать дату (в формате строки `String`), либо считывать дату (из строки `String`). Выглядит это всё следующим образом:

```
1  Date date = new Date(); // получаем текущую дату
2  SimpleDateFormat formatter = new SimpleDateFormat("d-MM-yy HH:mm:ss"); //создаём экземпляр класса
3                                          //и передаём ему шаблон представления даты и времени
4  String dateAsString = formatter.format(date); //преобразуем дату в строку заданного формата
5
6  Date dateAfterConversion = formatter.parse(dateAsString); //преобразуем строку обратно в дату
```

8. **URI** расшифровывается как Uniform Resource Identifier и переводится как "унифицированный идентификатор ресурса". URI — это последовательность символов, идентифицирующая абстрактный или физический ресурс. URL расшифровывается как Uniform Resource Locator. То есть это некий унифицированный указатель на ресурс, однозначно определяющий его месторасположение. URL служит стандартизированным способом записи адреса ресурса в сети Интернет.
   Их отличия в том, что **URI** — это некоторый идентификатор ресурса, который позволяет этот ресурс как-то идентифицировать, а **URL** — это указатель на ресурс, он даёт информацию о том, где находится ресурс. Таким образом URL — это URI, который помимо идентификации ресурса, даёт информацию о его местонахождении.

9. **Сокеты** — это связка *IP-адрес + порт*, позволяющая из внешней сети однозначно идентифицировать программу на компьютере или сервере. В Java для работы с сокетами есть два класса `Socket` и `ServerSocket`. Экземпляры первого класса играют роль клиента, экземпляры второго — роль сервера. Клиент может отправлять и принимать сообщения через сокет. Сервер же постоянно отслеживает запросы пользователей и отвечает на них.
   Для того, чтобы отправить данные через сокет, в классе `Socket` существует класс `getOutnputStream()`, возвращающий исходящий поток, с которым уже можно работать как обычно. Для приёма информацию нужно воспользоваться методом `getInputStream()`, который возвращает входящий поток. Дальше с этим потоком можно работать как с обычно потом ввода. Также стоит отметить, что при создании клиентского сокета (экземпляра класса `Socket`) в конструктор нужно передать ip-адрес сервера и порт, на котором он работает принимающая программа-сервер.
   При создании серверного сокета (экземпляра класса `ServerSocket`) нужно указывать только порт, через который будет работать программа. После этого вызывается метод `accept()`. Этот метод ожидание подключение клиента, а после этого возвращает экземпляр класса `Socket`, необходимый для взаимодействия с этим клиентом. Дальше работать идёт с экземпляром класса `Socket`, как в первом случае (в случае клиента).

10. Главное отличие в том, что класс **URL** предназначен для работы с URL-строкой (парсинг URL-строки), а **Socket** используется для соединения с удалённым сервером и отправки информации на сервер и/или приёма информации от сервера (хотя, используя класс URL, можно получить доступ к ресурсу, на который указывает сам URL; но делается это не напрямую, а через объект класса URLConnection). Также, если смотреть в общем, то Socket используется для связи с сервером (другой программой), а URL — для доступа к ресурсу (например, к файлу). Кроме того, URL и URLConnection ориентированы в основном на работу с HTTP, тогда как Socket может работать с любыми протоколами.

lichMax

+45

Комментарии (6) популярные новые старые

JavaCoder

Введите текст комментария

barracuda Уровень 41, Санкт-Петербург, Россия EXPERT 12 апреля 2021, 10:17

Где народ? Где комменты?

Ответить +1

Даниил Salesforce Developer в **Customertimes** MASTER 1 ноября 2019, 01:18

Ответ на 2-й вопрос, только по понятнее. И тут не плохо рассказано.

Ответить +2

Алексей Бутузов Уровень 38, Россия 9 февраля 2019, 21:50

Немчинов Сергей40 уровень, Новосибирск
16 декабря 2017, 16:05
> В чем отличие URI и URL?

Здесь есть отличное описание:
https://habrahabr.ru/post/190154/

Ответить +3

Даниил Salesforce Developer в **Customertimes** MASTER 31 октября 2019, 22:48

Тут тоже. Спасибо тому кто это выложил)

Ответить +3

Interstellar Java Developer в **EPAM** EXPERT 8 сентября 2020, 15:12

А здесь ещё лучше

Ответить 0

Артём Кашкин Уровень 41, Харьков, Украина 14 октября 2018, 16:31

Спасибо за ответы !

Ответить +1

ПОДПИСЫВАЙТЕСЬ

ЯЗЫК ИНТЕРФЕЙСА

Русский