Екзамен ООП 2 семестр

Мережеві протоколи та технології

1. **Що таке мережеві протоколи? Для чого вони використовуються? В чому відмінність між мережевими протоколами, програмами, застосунками? Навести приклади популярних протоколів.**

Мережеві протоколи - це набір правил і стандартів, які визначають, як пристрої в мережі обмінюються даними. Вони встановлюють формат повідомлень, порядок їх передачі, обробку помилок і контроль потоку. Протоколи використовуються для забезпечення взаємодії між різними пристроями, стандартизації передачі даних, гарантування надійності зв'язку та централізованого управління мережею.

Відмінність між поняттями полягає в тому, що протокол це набір правил для комунікації, програма являє собою конкретну реалізацію функціональності, а застосунок - це програмне забезпечення для кінцевого користувача. Популярні протоколи включають HTTP/HTTPS для веб-комунікації, TCP/UDP для транспортного рівня, IP для мережевого рівня, FTP для передачі файлів, SMTP для електронної пошти та DNS для розв'язання доменних імен.

2. **Що таке стек протоколів? Описати два найбільш популярні стеки протоколів, їх рівні, приклади протоколів на кожному рівні. Порівняти ці стеки. Який з них є найбільш популярним в сучасних мережах, чому?**

Стек протоколів - це ієрархічна структура протоколів, де кожен рівень надає послуги верхньому рівню. Найпопулярнішими є стеки TCP/IP та OSI.

TCP/IP складається з чотирьох основних рівнів: канальний рівень з протоколами Ethernet та ARP, мережевий рівень з IP та ICMP, транспортний рівень з TCP та UDP, та прикладний рівень з HTTP, FTP, SMTP і DNS. OSI модель має сім рівнів: фізичний рівень для електричних сигналів, канальний для Ethernet і Wi-Fi, мережевий для IP і IPX, транспортний для TCP і UDP, сеансовий для NetBIOS, рівень представлення для SSL і ASCII, та прикладний для HTTP і SMTP.

Порівнюючи ці стеки, TCP/IP практично використовується в Інтернеті та має чотири рівні, тоді як OSI є теоретичною моделлю для навчання з сімома рівнями. TCP/IP є найпопулярнішим у сучасних мережах, оскільки це основа всього Інтернету та він простіший у реалізації.

3. **Що таке пакети та подібні сутності в мережевих протоколах? Навіщо потрібно розбиття на пакети? Яка структура пакетів в популярних протоколах? Як називаються подібні сутності в різних протоколах?**

Пакети - це структуровані блоки даних, які передаються по мережі. Розбиття на пакети необхідне для ефективного використання мережі, можливості паралельної передачі, обробки помилок на рівні окремих пакетів та балансування навантаження.

Структура пакетів зазвичай включає заголовок із службовою інформацією, корисне навантаження з основними даними та опціональну кінцівку з контрольною сумою. Подібні сутності в різних протоколах називаються по-різному: в IP це пакет, у TCP це сегмент, в Ethernet це кадр, а в HTTP це повідомлення.

4. **Що таке адресація в мережевих протоколах? Які механізми адресації використовуються в різних протоколах, на різних рівнях моделі? Навести приклади правильних та неправильних адрес для кількох популярних протоколів (на різних рівнях).**

Адресація - це система ідентифікації пристроїв у мережі. Механізми адресації використовуються на різних рівнях моделі. На фізичному рівні використовуються 48-бітні MAC-адреси, на мережевому рівні застосовуються 32-бітні IPv4 адреси та 128-бітні IPv6 адреси, на транспортному рівні використовуються порти, а на прикладному рівні застосовуються URL адреси та email адреси.

Приклади правильних адрес включають IPv4 адреси як 192.168.1.1 чи 10.0.0.1, IPv6 адреси як ::1 чи fe80::1, та MAC адреси як AA:BB:CC:DD:EE:FF. Неправильні адреси можуть бути такими: IPv4 256.256.256.256 або неповна 192.168.1, IPv6 з неправильними символами gggg::1, або MAC адреса з неіснуючими символами ZZ:BB:CC:DD:EE:FF.

5. **Що таке протоколи IP, TCP, UDP? Для чого вони використовуються? Які їх особливості? Який формат даних/пакетів, яким чином відбувається мережева взаємодія? Навести приклади застосунків, які їх використовують.**

IP (Internet Protocol) призначений для маршрутизації пакетів між мережами. Він працює без гарантії доставки та без встановлення з'єднання. Формат

включає заголовок IP плюс дані. IP є основою для всього інтернет-трафіку та використовується практично всіма мережевими застосунками.

TCP (Transmission Control Protocol) забезпечує надійну передачу даних. Він є з'єднання-орієнтованим протоколом, що гарантує доставку, контролює потік та виявляє й виправляє помилки. Формат складається з заголовка TCP та даних. TCP використовується такими застосунками як HTTP, HTTPS, FTP та SMTP.

UDP (User Datagram Protocol) призначений для швидкої передачі без гарантій. Він працює без встановлення з'єднання, не гарантує доставку та має мінімальний заголовок лише 8 байт. UDP використовується для DNS запитів, DHCP, онлайн ігор та потокового відео.

HTTP протокол

6. **Що таке протокол HTTP? Для чого він використовується? Які його особливості? Який формат даних/пакетів, яким чином відбувається мережева взаємодія? Навести приклади застосунків, які його використовують.**

HTTP (HyperText Transfer Protocol) - це протокол прикладного рівня для передачі гіпертекстових документів. Він використовується веб-браузерами та веб-серверами, для API комунікації та веб-сервісів. HTTP є протоколом без збереження стану, використовує текстовий формат, працює за моделлю запит-відповідь та зазвичай використовує TCP порт 80.

Формат HTTP повідомлення включає рядок запиту з методом, URL та версією, заголовки для метаданих, порожній рядок як роздільник та опціональне тіло з даними. HTTP застосовується у веб-браузерах як Chrome та Firefox, мобільних додатках, REST API та системах веб-скрейпінгу.

7. **Що таке запити і відповіді HTTP? Що таке заголовки HTTP? Навести приклади популярних заголовків для запитів та відповідей HTTP.**

HTTP запит складається з рядка запиту, що містить метод, URL та версію протоколу, заголовків з метаданими, порожнього рядка як роздільника та опціонального тіла з даними. HTTP відповідь включає рядок статусу з версією, кодом та повідомленням, заголовки, порожній рядок та тіло відповіді.

Популярні заголовки запитів включають Host для вказання домену, User-Agent для ідентифікації клієнта, Accept для вказання бажаних типів контенту, Authorization для автентифікації, Content-Type для типу даних та Cookie для збереження стану. Популярні заголовки відповідей містять Content-Type для типу контенту, Content-Length для розміру, Set-Cookie для встановлення cookies, Location для перенаправлень, Cache-Control для кешування та Server для ідентифікації сервера.

8. **Що таке методи (дієслова) в HTTP? Навести найбільш популярні, описати для чого вони використовуються, порівняти їх між собою. Навести приклади запитів та відповідей для популярних методів, для популярних сценаріїв використання.**

GET використовується для отримання ресурсу, є безпечним та ідемпотентним методом з параметрами в URL. POST призначений для створення ресурсу, не є

ідемпотентним та передає дані в тілі запиту. PUT служить для оновлення або створення ресурсу та є ідемпотентним. DELETE використовується для видалення ресурсу та також є ідемпотентним.

PATCH призначений для часткового оновлення ресурсу. HEAD працює як GET, але повертає тільки заголовки, що корисно для перевірки існування ресурсу. OPTIONS надає інформацію про доступні методи та використовується для CORS preflight запитів.

Приклади використання включають GET /users/123 для отримання користувача, POST /users з JSON даними для створення, PUT /users/123 для повного оновлення, DELETE /users/123 для видалення та PATCH /users/123 для часткового оновлення.

9. **Що таке коди відповідей в HTTP? Для чого вони використовуються? Навести приклади найбільш популярних, в яких ситуаціях їх доцільно використовувати.**

Коди відповідей HTTP групуються за категоріями. 1xx коди є інформаційними, наприклад 100 Continue вказує продовжити запит. 2xx коди означають успіх: 200 OK для успішного виконання, 201 Created для створення ресурсу, 204 No Content для успішного виконання без контенту.

3xx коди вказують на перенаправлення: 301 Moved Permanently для постійного перенаправлення, 302 Found для тимчасового, 304 Not Modified коли ресурс не змінено. 4xx коди означають помилки клієнта: 400 Bad Request для неправильного запиту, 401 Unauthorized для неавторизованого доступу, 403 Forbidden для забороненого доступу, 404 Not Found коли ресурс не знайдено, 409 Conflict для конфлікту даних.

5xx коди вказують на помилки сервера: 500 Internal Server Error для внутрішніх помилок, 502 Bad Gateway для проблем зі шлюзом, 503 Service Unavailable коли сервіс недоступний.

Безпека

10. **Які механізми безпеки існують в популярних мережевих протоколах? Порівняти їх. Які типові атаки на мережеві протоколи, яким чином механізми безпеки протидіють цим атакам?**

Основні механізми безпеки включають шифрування симетричне як AES, асиметричне як RSA та гібридне як TLS. Автентифікація забезпечується цифровими сертифікатами, токенами доступу та біометричними даними. Контроль цілісності здійснюється через хеш-функції як SHA-256 та коди автентифікації повідомлень.

Типові атаки включають man-in-the-middle атаки, DDoS атаки, spoofing та replay атаки. Протидія цим атакам здійснюється через TLS/SSL для шифрування, firewall для фільтрації трафіку, системи виявлення вторгнень IDS/IPS та обмеження швидкості запитів.

11. **Що таке HTTPS? Описати як це працює, особливості реалізації, налаштування. Від яких атак захищає HTTPS?**

HTTPS - це HTTP over SSL/TLS, що забезпечує безпечну передачу даних.
Процес роботи включає підключення клієнта до сервера через порт 443, TLS handshake з обміном сертифікатами, встановлення симетричного ключа та подальшу шифровану комунікацію.

Особливості реалізації включають необхідність SSL-сертифіката, використання порту 443 та додаткове навантаження на процесор. Налаштування потребує отримання сертифіката від центру сертифікації як Let's Encrypt, конфігурації веб-сервера та налаштування редиректу з HTTP на HTTPS. HTTPS захищає від man-in-the-middle атак, підслуховування та підміни даних.

Архітектури

12. **Що таке архітектура клієнт-сервер? Описати основні компоненти, їх взаємодію. Переваги та недоліки цієї архітектури. Приклади застосунків, що її використовують.**

Архітектура клієнт-сервер складається з основних компонентів: клієнта як ініціатора запитів, сервера як обробника запитів та мережі як каналу зв'язку. Взаємодія відбувається таким чином: клієнт відправляє запит, сервер обробляє запит, сервер повертає відповідь, клієнт обробляє отриману відповідь.

Переваги архітектури включають централізоване управління даними, покращену безпеку, масштабованість та консистентність даних. Недоліки містять наявність єдиної точки відмови, залежність від мережі та можливі вузькі місця продуктивності. Приклади використання включають веб-браузери з веб-серверами, email клієнти з поштовими серверами та мобільні додатки з API серверами.

13. **Що таке багаторівнева архітектура? Описати основні компоненти, їх взаємодію. Переваги та недоліки цієї архітектури. Приклади застосунків, що її використовують.**

Багаторівнева архітектура складається з трьох основних рівнів. Рівень представлення відповідає за користувацький інтерфейс та включає веб-браузери й мобільні додатки. Рівень бізнес-логіки обробляє бізнес-правила та працює на application серверах. Рівень даних відповідає за зберігання даних в базах даних.

Взаємодія відбувається через канали: презентація взаємодіє з логікою, логіка взаємодіє з даними. Переваги включають модульність системи, можливість незалежної розробки компонентів, повторне використання коду та покращену безпеку. Недоліки містять збільшену складність системи, мережеві затримки та більше точок потенційної відмови. Приклади застосування включають корпоративні системи, e-commerce платформи та ERP системи.

Веб-технології

14. **Описати різні схеми організації вебзастосунків - класичні (Web 1.0), AJAX, SPA, SSR. Описати типову взаємодію між клієнтом та сервером для кожної з них. Порівняти за можливостями, за складністю реалізації на сервері та клієнті.**

Web 1.0 використовує статичні HTML сторінки з повним перезавантаженням сторінки та серверним рендерингом, що забезпечує простоту та SEO-friendly контент. AJAX дозволяє асинхронні запити з частковим оновленням сторінки через JavaScript XMLHttpRequest, покращуючи користувацький досвід.

SPA (Single Page Application) працює як одна HTML сторінка з динамічним контентом через JavaScript, використовує client-side routing та фреймворки як React, Angular, Vue. SSR (Server-Side Rendering) виконує рендеринг на сервері для швидкого первинного завантаження та SEO оптимізації, використовуючи технології як Next.js та Nuxt.js.

Порівняння за складністю реалізації показує, що на сервері складність зростає від Web 1.0 до SSR, AJAX та SPA, тоді як на клієнті складність збільшується від Web 1.0 до AJAX, SSR та SPA.

15. **Що таке вебфреймворки? Які основні можливості вони надають? Навести приклади популярних фреймворків дл різних мов програмування.**

Веб-фреймворки надають основні можливості для розробки: маршрутизацію для обробки URL запитів, шаблонізацію для генерації HTML, роботу з базами даних через ORM, автентифікацію та авторизацію користувачів, middleware для обробки запитів та системи безпеки.

Приклади популярних фреймворків включають для Java: Spring Framework, Struts та JSF; для Python: Django, Flask та FastAPI; для JavaScript: Express.js для Node.js, React, Angular та Vue для frontend; для PHP: Laravel, Symfony та CodeIgniter; для C#: ASP.NET Core та ASP.NET MVC.

16. **Що таке URL? Як описується відображення - які фрагменти коду будуть обробляти які URL? Порівняти різні способи такого відображення.**

URL (Uniform Resource Locator) - це адреса ресурсу в мережі зі структурою https://example.com:8080/path/to/resource?param=value#fragment.

Відображення URL на код може бути статичним, де /users відображається на UsersController, або динамічним, де /users/{id} відображається на UsersController.getUser(id).

Способи відображення включають використання конфігураційних файлів як web.xml, анотації як @WebServlet та @RequestMapping, та підхід "convention over configuration" де структура URL автоматично відображається на код.

17. Як відбувається взаємодія між клієнтом і сервером у вебзастосунках? Описати основні методи взаємодії, їх переваги та недоліки.

Основні методи взаємодії включають синхронну взаємодію, де клієнт чекає відповіді через блокуючі запити, що забезпечує простоту реалізації. Асинхронна взаємодія використовує AJAX запити з callbacks та promises, не блокуючи користувацький інтерфейс. Real-time взаємодія застосовує WebSockets, Server-Sent Events та push-нотифікації для миттєвого обміну даними.

Переваги та недоліки різних підходів: синхронна взаємодія проста в реалізації, але блокує UI; асинхронна забезпечує responsiveness, але складніша в реалізації; real-time надає миттєвість, але є ресурсоємною.

18. Які проблеми можливі з оновленням сторінки при використанні POST запитів? Які є методи вирішення цих проблем? Порівняти ці методи.

Проблеми з використанням POST включають повторну відправку форми при оновленні сторінки, дублювання даних та небажані побічні ефекти. Основним методом вирішення є PRG (Post-Redirect-Get) патерн, де після POST запиту сервер відправляє 302 Redirect на GET сторінку.

Інші методи включають використання CSRF токенів як унікальних токенів для кожної форми з перевіркою на сервері, та JavaScript обробку з відключенням кнопки після відправки та показом індикатора завантаження. Порівняння методів показує, що PRG забезпечує найкращий UX та є стандартним підходом, CSRF токени покращують безпеку, а JavaScript рішення надають швидкість, але потребують підтримки JavaScript.

Автентифікація та авторизація

Автентифікація - це перевірка особи користувача, тоді як авторизація - це перевірка прав доступу користувача до ресурсів.

19. Що таке автентифікація, авторизація? В чому між ними відмінність? Як це реалізується в вебзастосунках? Описати детально cookies, сесії.

Cookies та сесії працюють таким чином: cookie зберігає session ID на клієнті, сесія на сервері містить дані користувача, cookies автоматично відправляються з кожним запитом. Це забезпечує збереження стану між запитами.

20. Що таке автентифікація, авторизація? В чому між ними відмінність? Як це реалізується в вебзастосунках? Описати детально JWT токени.

JWT (JSON Web Tokens) - це самодостатні токени зі структурою header.payload.signature, що дозволяють роботу без збереження стану на сервері. Токен містить всю необхідну інформацію про користувача та його права.

21. Що таке автентифікація, авторизація? В чому між ними відмінність? Як це реалізується в вебзастосунках? Описати детально OAuth.

OAuth - це протокол авторизації для делегування доступу, що використовує authorization code flow. Популярні провайдери включають Google, Facebook, GitHub. OAuth дозволяє застосункам отримувати обмежений доступ до ресурсів користувача без отримання його облікових даних.

Основи сервлетів

22. Що таке життєвий цикл сервлета? Чим він відрізняється від життєвого циклу об'єкта? Навести приклади з власного коду.

Життєвий цикл сервлета включає чотири основні етапи: завантаження класу сервлета контейнером, ініціалізацію з викликом методу init(), обслуговування запитів через виклик service() та знищення з викликом destroy().

Відмінності від життєвого циклу звичайного об'єкта полягають у тому, що сервлет керується контейнером сервлетів, існує один екземпляр на весь додаток, ініціалізація відбувається лише один раз при першому запиті або при старті додатку.

23. Як будувати та розгортати вебзастосунки на основі сервлетів? В чому відмінність від просто Java застосунків? від вебзастосунків іншими мовами чи з використанням інших технологій? Навести приклади з власного коду.

Будування веб-застосунків включає компіляцію Java класів, упакування в WAR файл, включення всіх залежностей та створення конфігурації web.xml. Розгортання відбувається через копіювання WAR файлу в директорію webapps application сервера, автоматичне розпакування та запуск контейнера сервлетів.

Відмінності від звичайних Java застосунків включають необхідність веб-контейнера для виконання, використання WAR формату замість JAR, наявність веб-специфічних конфігурацій та залежність від servlet API. Відмінності від веб-застосунків на інших мовах включають компіляцію в байт-код, використання JVM та специфічну архітектуру Java EE.

24. Як конфігурувати вебзастосунки на основі сервлетів? Навести два основних способи, порівняти їх. Навести приклади з власного коду.

Існують два основні способи конфігурації веб-застосунків. XML конфігурація використовує файл web.xml для централізованого налаштування сервлетів, фільтрів, слухачів та інших компонентів. Конфігурація через анотації дозволяє налаштовувати компоненти безпосередньо в коді класів.

XML конфігурація забезпечує централізоване управління та відокремлення конфігурації від коду, що полегшує зміни без перекомпіляції. Анотації надають

зручність розробки та близькість конфігурації до коду, але можуть ускладнити централізоване управління.

25. Що таке сервлети? Навіщо вони використовуються? В чому відмінність між Servlet API, J2EE, Jakarta? Навести приклади з власного коду.

Сервлети - це Java класи для обробки HTTP запитів у веб-додатках. Вони використовуються для динамічної генерації веб-контенту, створення API endpoints, обробки форм та взаємодії з базами даних.

Відмінності між API включають: Servlet API - це оригінальний API від Sun/Oracle для створення веб-компонентів; J2EE (Java 2 Enterprise Edition) - старша назва для набору enterprise технологій Java; Jakarta - нова назва після передачі проекту від Oracle до Eclipse Foundation, що включає оновлені пакети та простори імен.

26. Що таке сервлети? Навіщо вони використовуються? В чому відмінність між сервлетом, класом, об'єктом, методом, сервером, сервісом, мікросервісом? Навести приклади з власного коду.

Сервлет vs Клас: сервлет є спеціалізованим Java класом, що наслідує HttpServlet, має специфічні методи для обробки HTTP запитів та керується контейнером сервлетів.

Сервлет vs Об'єкт: зазвичай існує один екземпляр сервлета на весь додаток, він обробляє запити в багатопоточному режимі та має специфічний життєвий цикл.

Сервлет vs Метод: сервлет містить методи для обробки різних типів HTTP запитів, має власний життєвий цикл та може зберігати стан між викликами.

Сервлет vs Сервер: сервлет працює всередині веб-сервера або application сервера, сервер керує життєвим циклом сервлетів та надає їм runtime середовище.

Сервлет vs Сервіс: сервлет є веб-компонентом для обробки HTTP запитів, тоді як сервіс містить бізнес-логіку та може використовуватися різними компонентами.

Сервлет vs Мікросервіс: мікросервіс є архітектурним патерном для побудови розподілених систем, може містити сервлети як компоненти для обробки запитів, але представляє собою повноцінний автономний сервіс.

JSP

27. Що таке JSP? Для чого воно використовується? Чи можна (та чи варто) реалізувати повноцінний вебзастосунок на основі сервлетів без використання JSP? Навести приклади з власного коду.

JSP - це технологія для створення динамічних веб-сторінок, що дозволяє змішувати HTML з Java кодом. JSP використовується для шаблонізації, відображення даних з серверної частини та створення користувацького інтерфейсу веб-додатків.

Можна реалізувати повноцінний веб-застосунок тільки на сервлетах без JSP, але це буде незручно для складного HTML, потребуватиме багато коду для генерації розмітки та ускладнить підтримку. JSP значно спрощує створення динамічних сторінок та розділяє логіку від представлення.

28. Як реалізувати стандартні конструкції керування (опціональні елементи, альтернативи, цикли) з використанням JSP? Навести різні варіанти реалізації, порівняти їх. Навести приклади з власного коду.

Для реалізації стандартних конструкцій керування в JSP існують різні підходи. Scriptlets дозволяють писати Java код безпосередньо в JSP, але це вважається поганою практикою. JSTL (JSP Standard Tag Library) надає спеціальні теги для керування потоком виконання. Expression Language (EL) спрощує доступ до даних та їх відображення.

JSTL з EL є найкращим підходом, оскільки забезпечує чистіший код, кращу читабельність, менше Java коду в представленні та легшу підтримку. Scriptlets слід уникати, оскільки вони змішують логіку з представленням та ускладнюють підтримку.

Класи та методи

29. **Що таке клас HttpServlet? Які в ньому основні методи? Як його використовувати? Навести приклади з власного коду.**

HttpServlet - це абстрактний клас, що є основою для створення HTTP сервлетів. Основні методи включають doGet() для обробки GET запитів, doPost() для POST запитів, doPut() для PUT запитів, doDelete() для DELETE запитів, doHead() для HEAD запитів та doOptions() для OPTIONS запитів. Також є методи init() для ініціалізації, destroy() для очищення ресурсів та service() для загальної обробки запитів.

HttpServlet використовується через наслідування та перевизначення необхідних методів. Контейнер сервлетів автоматично викликає відповідний метод залежно від типу HTTP запиту.

30. **Що таке контейнер сервлетів? Навіщо вони використовуються? В чому відмінність між контейнерами сервлетів, IoC containers, контейнерами (колекціями) зі стандартної бібліотеки Java? Навести приклади з власного коду.**

Контейнер сервлетів - це runtime середовище, що керує життєвим циклом сервлетів, обробляє HTTP запити, надає servlet API та забезпечує безпеку й управління ресурсами.
Відмінність від IoC containers полягає в тому, що контейнер сервлетів спеціалізується на веб-компонентах та HTTP протоколі, тоді як IoC контейнери керують залежностями будь-яких об'єктів. Відмінність від колекцій Java в тому, що контейнер сервлетів активно керує об'єктами, а колекції лише зберігають дані.

31. **Що таке контейнер сервлетів? Навіщо вони використовуються? В чому відмінність між контейнерами сервлетів, web server, application server? Навести приклади з власного коду.**

Контейнер сервлетів - це runtime середовище, що керує життєвим циклом сервлетів, обробляє HTTP запити, надає servlet API та забезпечує безпеку й управління ресурсами.

Відмінність між servlet container, web server та application server: web server обслуговує статичний контент та HTTP запити; servlet container виконує

сервлети та JSP; application server включає servlet container плюс додаткові enterprise сервіси як EJB, JMS, JTA.

32. Що таке методи doGet() та doPost()? В чому між ними відмінність? Коли який варто використовувати? Навести приклади з власного коду.

doGet() призначений для безпечних операцій читання даних, є ідемпотентним, передає параметри через URL query string та використовується для отримання ресурсів. doPost() призначений для операцій, що змінюють стан сервера, не є ідемпотентним, передає дані в тілі запиту та використовується для створення, оновлення чи видалення ресурсів.

doGet слід використовувати для пошуку, фільтрації, сторінок з формами, навігації по сайту. doPost варто застосовувати для відправки форм, завантаження файлів, операцій аутентифікації, будь-яких операцій, що змінюють дані.

33. Що таке класи HttpServletRequest та HttpServletResponse? В чому між ними відмінність? До якого простору імен/пакету вони належать? Як їх використовувати? Навести приклади з власного коду.

HttpServletRequest інкапсулює всю інформацію про HTTP запит, що надходить від клієнта. Цей об'єкт надає доступ до параметрів запиту, заголовків, cookies, інформації про сесію, тіла запиту та метаданих про клієнта.

Основні можливості включають отримання параметрів через getParameter(), доступ до заголовків через getHeader(), роботу з cookies через getCookies(), управління сесіями через getSession(), отримання інформації про HTTP метод через getMethod(), доступ до URL компонентів та читання тіла запиту через getInputStream() або getReader().

HttpServletResponse представляє HTTP відповідь, яку сервер надсилає клієнту. Через цей об'єкт можна встановлювати статус відповіді, додавати заголовки, встановлювати cookies та записувати контент відповіді.
Основні можливості включають встановлення HTTP статусу через setStatus(), додавання заголовків через setHeader() та addHeader(), встановлення типу контенту через setContentType(), роботу з cookies через addCookie(), запис контенту через getWriter() або getOutputStream(), та перенаправлення через sendRedirect().

Основна відмінність полягає в напрямку потоку інформації: HttpServletRequest містить дані, що надходять від клієнта до сервера, тоді як HttpServletResponse

використовується для формування даних, що відправляються від сервера до клієнта. Request об'єкт є read-only для більшості операцій, крім атрибутів, тоді як Response об'єкт призначений для запису та налаштування відповіді.

Робота з базами даних

34. Як працювати з реляційною БД в застосунках на основі сервлетів? Як робити запити до БД, як виконувати основні CRUD операції? Навести приклади з власного коду.

Робота з реляційними базами даних в сервлетах здійснюється через JDBC API. Основні CRUD операції включають Create для вставки нових записів, Read для вибірки даних, Update для оновлення існуючих записів та Delete для видалення записів.

Для роботи з БД потрібно встановити з'єднання, підготувати SQL запити, виконати їх та обробити результати. Важливо правильно керувати ресурсами та обробляти виключення.

35. Як тестувати роботу з БД в застосунках но основі сервлетів? Навести приклади з власного коду.

Тестування роботи з базою даних в сервлетах включає використання тестової бази даних, моки об'єктів, інтеграційні тести та unit тести. Основні підходи включають використання in-memory баз даних як H2, контейнерного тестування з Testcontainers та моків для ізоляції компонентів.

Сесії та безпека

36. Як працювати з cookies та сесіями в вебзастосунках на основі сервлетів? Навести приклади з власного коду.

Cookies - це невеликі фрагменти даних, що зберігаються на клієнті та автоматично відправляються з кожним запитом. Сесії - це механізм збереження стану користувача на сервері, що ідентифікується через session ID в cookie або URL.

Cookies використовуються для збереження налаштувань користувача, запам'ятовування логіну, відстеження активності. Сесії застосовуються для автентифікації, збереження тимчасових даних, корзини покупок та іншої інформації про стан користувача.

37. Як працювати з JWT токенами в вебзастосунках на основі сервлетів? Навести приклади з власного коду.

JWT (JSON Web Tokens) - це самодостатні токени для передачі інформації між сторонами. JWT складається з трьох частин: header.payload.signature, де header містить тип токену та алгоритм, payload містить дані (claims), signature забезпечує цілісність.

JWT використовуються для автентифікації без збереження стану на сервері, авторизації з передачею ролей та прав, безпечного обміну інформацією між сервісами.

Веб-сторінки та ресурси

38. **Як реалізувати статичні (з точки зору сервера) вебсторінки чи ресурси в застосунках на основі сервлетів? Навести різні способи, порівняти їх. Навести приклади з власного коду.**

Статичні веб-сторінки - це контент, що не змінюється на сервері та віддається як є. Способи реалізації включають розміщення файлів у webapp директорії, використання DefaultServlet контейнера та налаштування статичних ресурсів через веб-сервер.

39. **Як реалізувати динамічні (з точки зору сервера) вебсторінки в застосунках на основі сервлетів? Навести різні способи, порівняти їх. Навести приклади з власного коду.**

Динамічні веб-сторінки генеруються на сервері на основі даних та логіки. Способи реалізації включають використання сервлетів для генерації HTML, JSP для шаблонізації, комбінування сервлетів з JSP та використання шаблонізаторів як Thymeleaf.

Сервери та контейнери

40. Що таке application server? Навіщо вони використовуються? Навести приклади популярних реалізацій, як їх встановлювати та налаштовувати, як запускати в них власні застосунки. Навести приклади з власного коду.

Application Server - це програмне середовище, що надає runtime платформу для enterprise додатків. Application server використовуються для запуску веб-додатків, управління ресурсами, забезпечення безпеки, балансування навантаження та інтеграції з enterprise системами.

Популярні реалізації включають Apache Tomcat як servlet container, WildFly та GlassFish як повноцінні application server, WebLogic та WebSphere як комерційні рішення. Встановлення включає завантаження дистрибутиву, розпакування, налаштування конфігураційних файлів. Запуск додатків здійснюється через копіювання WAR файлів в webapps директорію або через адміністративні інтерфейси.

41. Що таке application server? Навіщо вони використовуються? В чому відмінність між application server, web server, servlet container? Навести приклади з власного коду.

Application Server - це програмне середовище, що надає runtime платформу для enterprise додатків. Application server використовуються для запуску веб-додатків, управління ресурсами, забезпечення безпеки, балансування навантаження та інтеграції з enterprise системами.

Відмінності між компонентами: Web server обслуговує HTTP запити та статичний контент; Servlet container виконує сервлети та JSP, є частиною application server; Application server включає servlet container плюс додаткові enterprise сервіси як EJB, JMS, JTA, JNDI.

Інструменти розробки

42. Що таке менеджер залежностей, менеджер пакетів? В чому між ними відмінність? Які популярні засоби використовуються в Java? Як їх використовувати під час реалізації вебзастосунків на основі сервлетів? Навести приклади з власного коду.

Менеджер залежностей керує бібліотеками та їх версіями в проекті, автоматично завантажує потрібні JAR файли та розв'язує конфлікти версій. Менеджер пакетів встановлює та керує програмним забезпеченням на системному рівні.

В Java популярні засоби включають Maven для управління проектами та залежностями, Gradle як сучасну альтернативу з гнучкою конфігурацією, SBT для Scala проектів. Для веб-застосунків на сервлетах ці інструменти дозволяють автоматично завантажувати servlet API, JDBC драйвери, бібліотеки логування, створювати WAR файли та управляти профілями збірки.

Maven використовує декларативний підхід з XML конфігурацією, має великий репозиторій бібліотек та стандартну структуру проекту. Gradle пропонує програмний підхід з Groovy/Kotlin DSL, швидшу збірку та гнучкість конфігурації.