23.01.02

한달간 피드백

17일

2가지 임계영역에 대한 lock 구현 방식 뮤텍스, 세마포어

- Mutex(Mutual Exclutsion)
  - 임계 구역에 대한 프로세스 간의 시간이 겹치지 않도록 하는 방식
  - 2개 프로세스 간의 경쟁을 기반으로 함
  - Lock을 소유한 프로세스만이 임계 영역에 접근 가능
  - 프로세스 단위의 동기화 방식
- Semaphore
  - 시그널 매커니즘(임계 영역 전 wait(), 임계 영역 후 signal())을 통해, 프로세스 간 임계 영역을 사용하도록 한 방식
  - 2개 이상의 프로세스 간 경쟁을 기반으로 함
  - 시스템 범위에서 커널이 소유한 세마포어를 통해 임계 영역에 접근 가능
  - 기다리는 방식 - Busy Waiting(초기) → Block-Wakeup(현재)

18일

Http 0.9, 1.0 1.1 2.0

Http 0.9

- 1990
- html 문서 요청 - 응답(just GET, 다른 메서드 x)
- 헤더 (Host, User-Agent, Accept)

- 응답 시 http 버전 및 status code 포함
- 요청에 데이터(body) 포함 X

Http 1.0

- 1996
- HEAD, POST 추가
- http 헤더, 바디, 응답코드 추가됨.
- **비지속 연결**(Non-Persistent Connection) - TCP 세션 유지 X
  - 1GET - 1CONNECTION
- 단순히 open/operation/close을 통한 flow의 제한 → 네트워크 혼잡, disconnect 발생 → 서버에 재접속 시도 → 서버 과부하, 성능 저

Http 1.1

- 1999
- OPTION, PUT, DELETE, TRACE 추가
- **지속 연결**(Persistent Connection, Keep-alive) - TCP 세션 유지 - O
  - N GET - 1CONNECTION
  - 파이프 라이닝 - 커넥션 내 순차 요청-응답
    - Network Latency 줄임
  - HOL 문제
- Host Header - 버츄얼 호스팅(1IP - N DOMAIN)
- 강력한 인증 절차 → 프록시에서 사용자의 인증을 요구
  - proxy-authentication, proxy-authorization 헤더 추가

Http 2.0

- 2015
- Multiplexed Streams
  - Stream과 frame을 통한 데이터 병렬 전송 - HOL 해결

- Stream Prioritization | 서버의 리소스에 대한 우선 순위지정, 리소스 로드 문제 해결
- 기존 Plain Text(평문, 개행으로 구분) → 바이너리 포맷으로 인코딩된 Message, Frame
- Stream: 연결 내에서 전달되는 바이트의 양방향 흐름, 하나 이상의 메시지가 전달 가능
- Message: 논리적 요청 또는 응답 메시지에 매핑되는 프레임의 전체 시퀀스.
- Frame: HTTP/2에서 통신의 최소 단위. 각 최소 단위에는 하나의 프레임 헤더가 포함. HEADERS Type Frame, DATA Type Frame이 존재

20일

last-modified

- 서버 응답 시 추가하는 헤더 값으로, 리소스의 마지막 수정 날짜를 포함.

21일

volatile 특징

- 항상 스레드가 캐시 없이 메인 메모리 영역에서 값을 참조. → 가시성 문제 해결
- 원자적 연산에서의 동기화 보장 (ex a=1 읽기, b=false 저장, b=a -a의 입장에서는 읽기 연산)
- 비원자적 연산에 대해서는 동기화를 보장할 수 없다. (ex a = a+1 - 레지스터에서 총 3개의 연산)

  → 배타적 실행 제어를 위해서는 synchronized를 사용해야 한다. 혹은 Atomic 클래스 사용

24일

의존성 주입 - 자동•수동 각각 언제 사용해야 하는가

- 자동 - 반복적인 비지니스 로직, 요구사항 구현 시 사용. 문제 발생 위치를 찾기가 어렵지 않기 때문에 자동 설정에도 문제 발생하지 않는다

- 수동
  - 기술적인 지원을 해야 할 때, AOP를 사용할 때. ex) 로그, DB
  - 문제 발생 위치를 찾기가 어려운 부분(애플리케이션 전반에 걸쳐 사용되는 것)에 대해, 사용처를 명확히 밝히기 위해 사용.

25일

싱글톤 빈에 프로토타입 빈이 주입될 경우

- 싱글톤 빈에 프로토타입 빈이 주입될 경우, 프로토타입 빈이 계속 유지된다.(프로토타입 빈의 주소를 받았으므로)
  - 해결책 - DL(Dependency Lookup)

    1) 싱글톤 빈에 필드 주입으로 ApplicationContext를 받고, 이를 통해 프로토타입 빈을 요청 - 스프링 컨테이너에 종속적인 코드 되고, 테스트 어려움

    2) Provider - ObjectProvider<Type> → 옵셔널, 스트림 등 편의 기능 있음. 스프링 의존적임

    3) Provider - Provider<Type> → JSR-330 라이브러리 사용. 자바 표준

객체의 생성과 초기화는 분리하는 것이 좋다

- 지연 로딩을 활용 → 객체를 사용할 때, 초기화 로직이 실행되는 것이 효율적

스프링 컨테이너 및 빈(싱글톤)의 생명 주기

- 스프링 컨테이너 생성 → 설정 주입 → 스프링 빈 생성 → 의존 관계 주입 → 초기화 콜백 → 사용 → 소멸 전 콜백 → 스프링 종료