WebSocket 동시 접속 테스트

WebSocket 동시 접속 성능 테스트 결과

프로젝트: Pickers
목적: WebSocket 서버의 동시 접속 처리 능력 및 메모리 안정성 검증

---

1. 테스트 환경

서버: Spring Boot 3.3.6 + Java 17
WebSocket: Java-WebSocket 라이브러리
포트: 9000
메시지: Kafka를 통한 실시간 브로드캐스팅

클라이언트: Chrome 브라우저, JavaScript WebSocket API
측정 도구: VisualVM 2.1.10 (JVM Heap 메모리 모니터링)

특이사항:

- WebSocket 서버 정상 동작 확인을 위해 실시간 시세 페이지(/realData) 유지
- 이로 인해 기본 연결 1개가 포함되어 접속자 수 +1 표시 (11/10, 51/50, 101/100)
- 메모리 측정은 기본 연결 포함한 전체 연결 기준

---

2. 테스트 시나리오

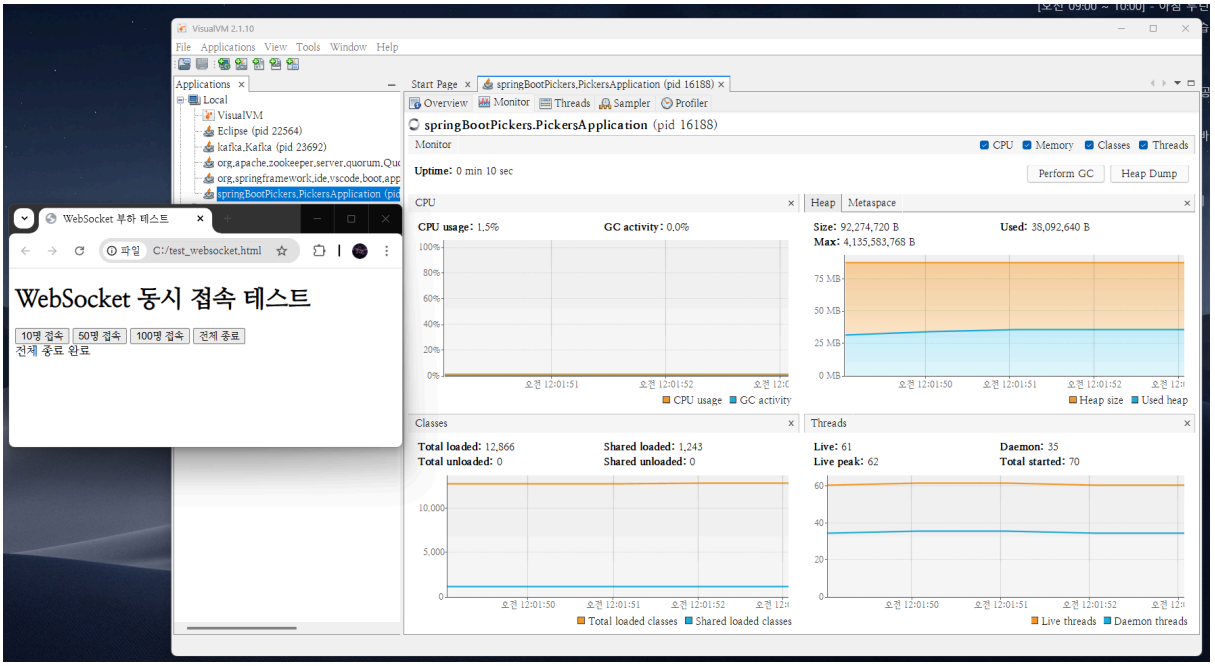
Step 1. 서버 실행 후 기본 메모리 측정
Step 2. 10명 동시 접속 후 메모리 측정
Step 3. 50명 동시 접속 후 메모리 측정
Step 4. 100명 동시 접속 후 메모리 측정
Step 5. 100명 동시 접속 후 30초 대기 → GC 동작 → 메모리 회수 확인

---

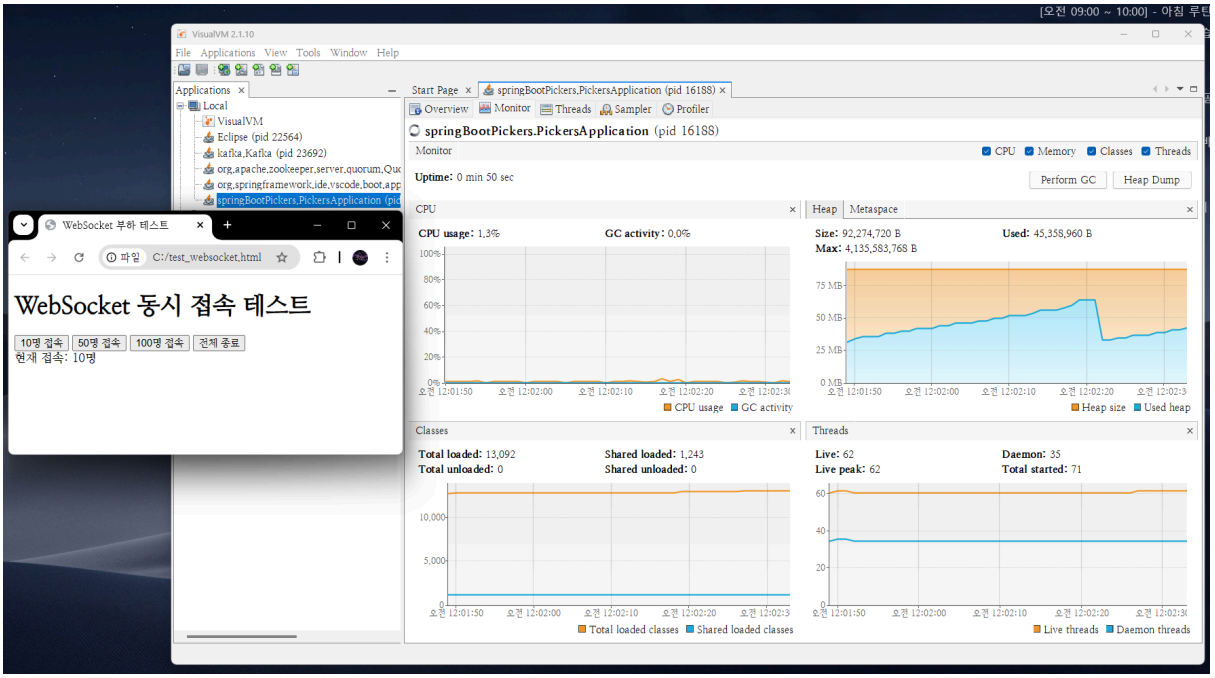
3. 측정 결과

3.1 메모리 사용량

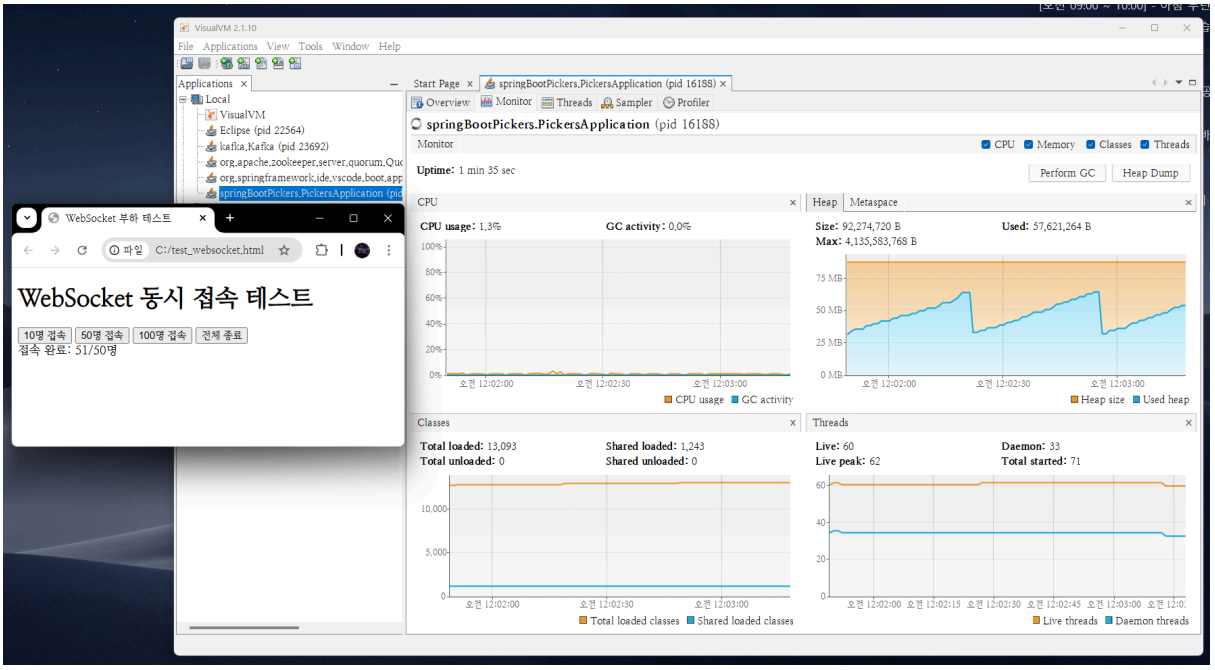
접속자 수	Used Heap	Heap Size	증가량
기본 (0명)	38 MB	88 MB	-
10명	45 MB	88 MB	+7 MB
50명	57 MB	88 MB	+19 MB
100명	74 MB	88 MB	+36 MB
GC 후	37 MB	88 MB	-37 MB



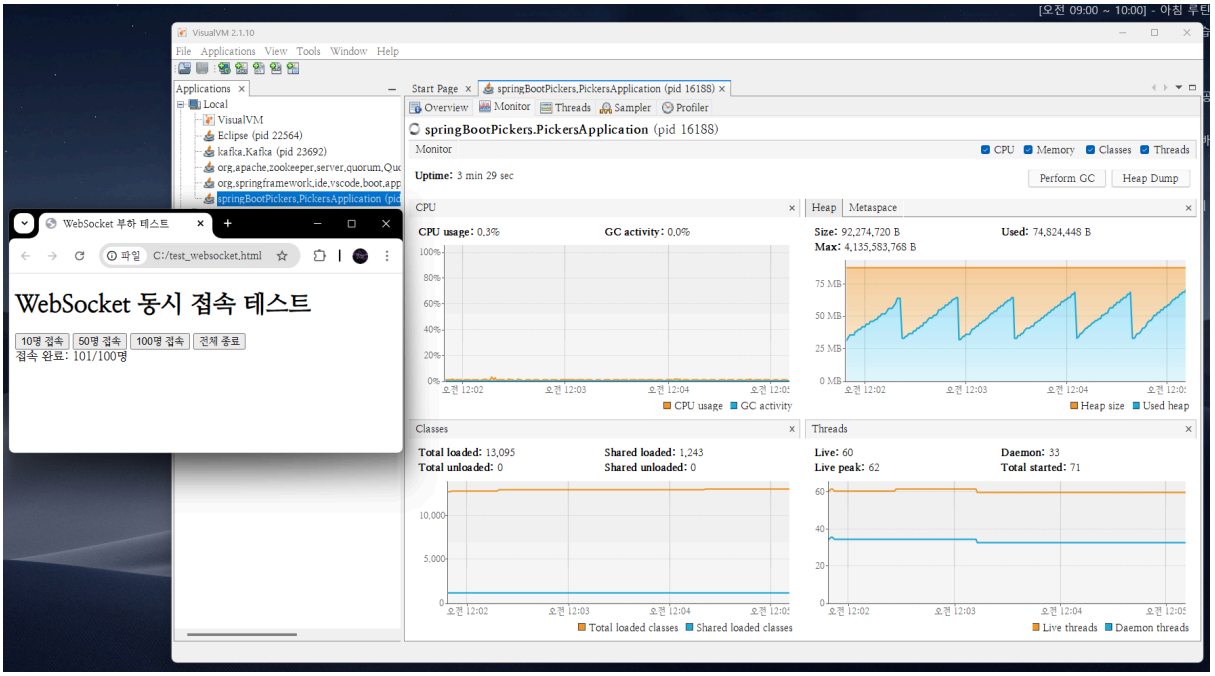
서버 실행 직후 - Used Heap 38MB



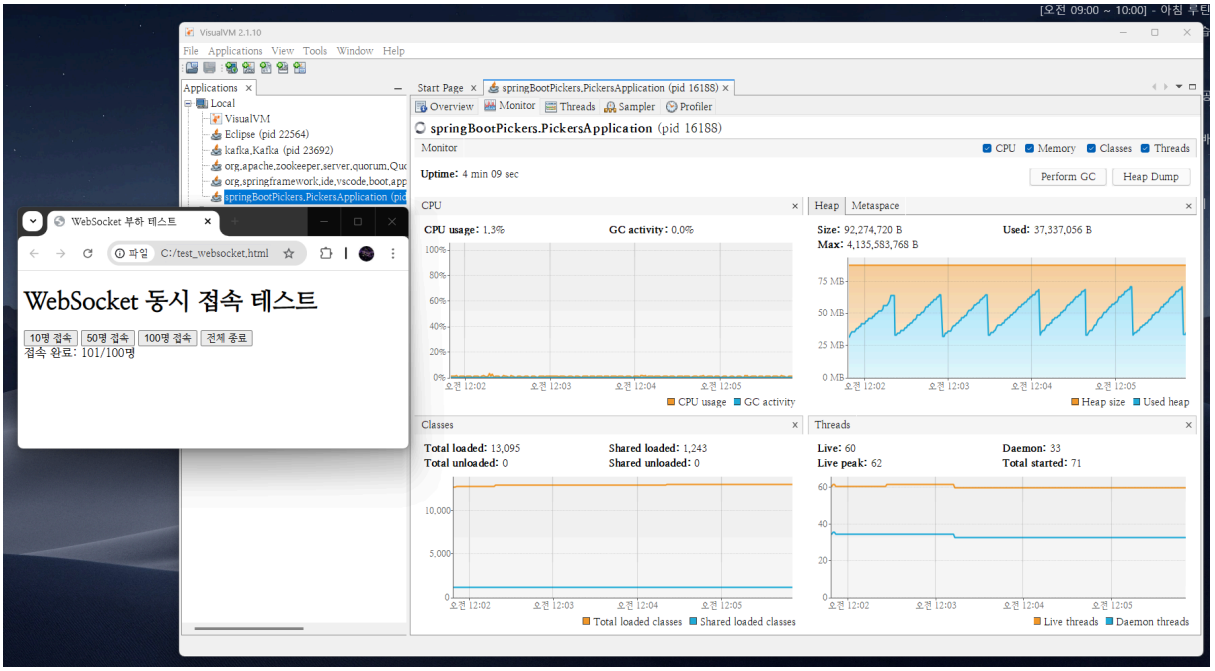
10명 접속 시 - Used Heap 45MB



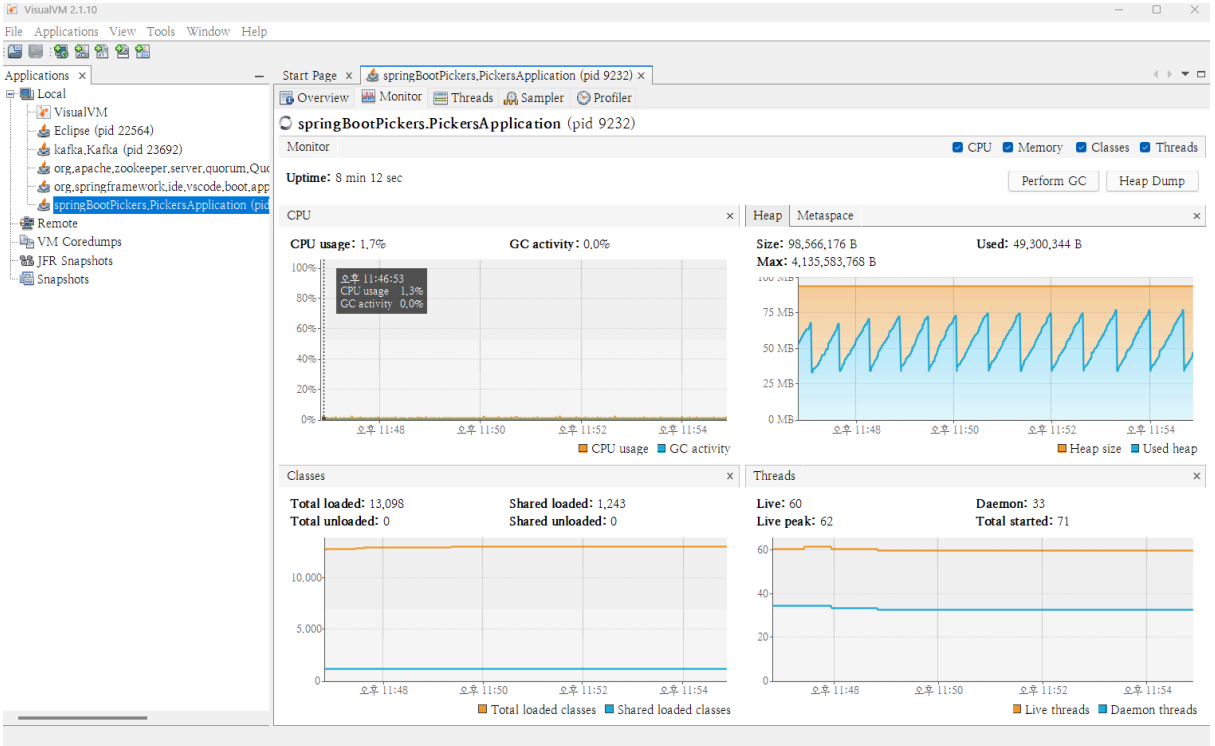
50명 접속 시 - Used Heap 57MB



100명 접속 시 - Used Heap 74MB



100명 접속 후 30초 대기 - Used Heap 37MB (GC 동작)



100명 접속 5분 후 - Used Heap 유지

3.2 접속자당 메모리 증가율

10명: 7 MB → 평균 0.7 MB/명
50명: 19 MB → 평균 0.38 MB/명
100명: 36 MB → 평균 0.36 MB/명

분석: 접속자가 증가할수록 평균 증가율 감소 (효율 증가)

3.3 VisualVM 상세 데이터

기본 (서버 실행 직후):

```
Used: 38,092,640 B (36.3 MB)
Size: 92,274,720 B (88.0 MB)
Max: 4,135,583,768 B (3.85 GB)
```

10명 접속:

```
Used: 45,358,960 B (43.3 MB)
Size: 92,274,720 B (88.0 MB)
접속 완료: 11/10명
```

50명 접속:

```
Used: 57,621,264 B (55.0 MB)
Size: 92,274,720 B (88.0 MB)
접속 완료: 51/50명
```

100명 접속:

```
Used: 74,824,448 B (71.4 MB)
Size: 92,274,720 B (88.0 MB)
접속 완료: 101/100명
```

100명 접속 후 30초 대기 (GC 동작):

```
Used: 37,337,056 B (35.6 MB)
Size: 92,274,720 B (88.0 MB)
결과: 기본 수준으로 복구
```

---

4. 메모리 회수 검증

4.1 GC 동작 확인

100명 접속 시: 74 MB
전체 종료 후: 37 MB (30초 대기)

회수량: 37 MB
회수율: 100% (기본 수준으로 복구)

4.2 메모리 누수 검증

결과:

- 100명 접속 → 종료 후 메모리가 기본 수준(38MB)으로 복구
- GC가 정상 동작하여 사용하지 않는 WebSocket 객체 회수
- ConcurrentHashMap의 onClose() 시 remove() 정상 작동

결론: 메모리 누수 없음

---

5. 기술적 분석

5.1 ConcurrentHashMap의 효과

코드:

```
private Set<WebSocket> connections = ConcurrentHashMap.newKeySet();

@Override
public void onClose(WebSocket conn, int code, String reason, boolean remote) {
    connections.remove(conn);  // 연결 해제 시 자동 제거
    log.debug("연결 해제: {}", conn.getRemoteSocketAddress());
}
```

효과:

- 스레드 안전한 연결 관리
- 연결 해제 시 자동으로 Set에서 제거
- 제거된 WebSocket 객체는 GC 대상

5.2 확장성 계산

현재 테스트 결과:

- 100명: 74 MB
- Max Heap: 3.85 GB

계산상 확장 가능 인원:

- 1,000명: 약 360 MB (100명 기준 선형 계산)
- 3,000명: 약 1.1 GB
- 10,000명: 약 3.6 GB (Max Heap 근처)

참고: 실제 테스트는 100명까지만 수행. 위 수치는 계산상 예상값.

---

6. 결론

측정 결과:

- 100명 동시 접속 시 메모리 74 MB로 안정적 유지
- 접속자당 평균 0.36 MB 소비
- GC 후 메모리 100% 회수 → 메모리 누수 없음

기술적 검증 완료:

1. ConcurrentHashMap으로 연결 관리
2. onClose() 시 자동 제거 동작
3. GC를 통한 메모리 회수 확인
4. 실시간 메시지 브로드캐스팅 안정성