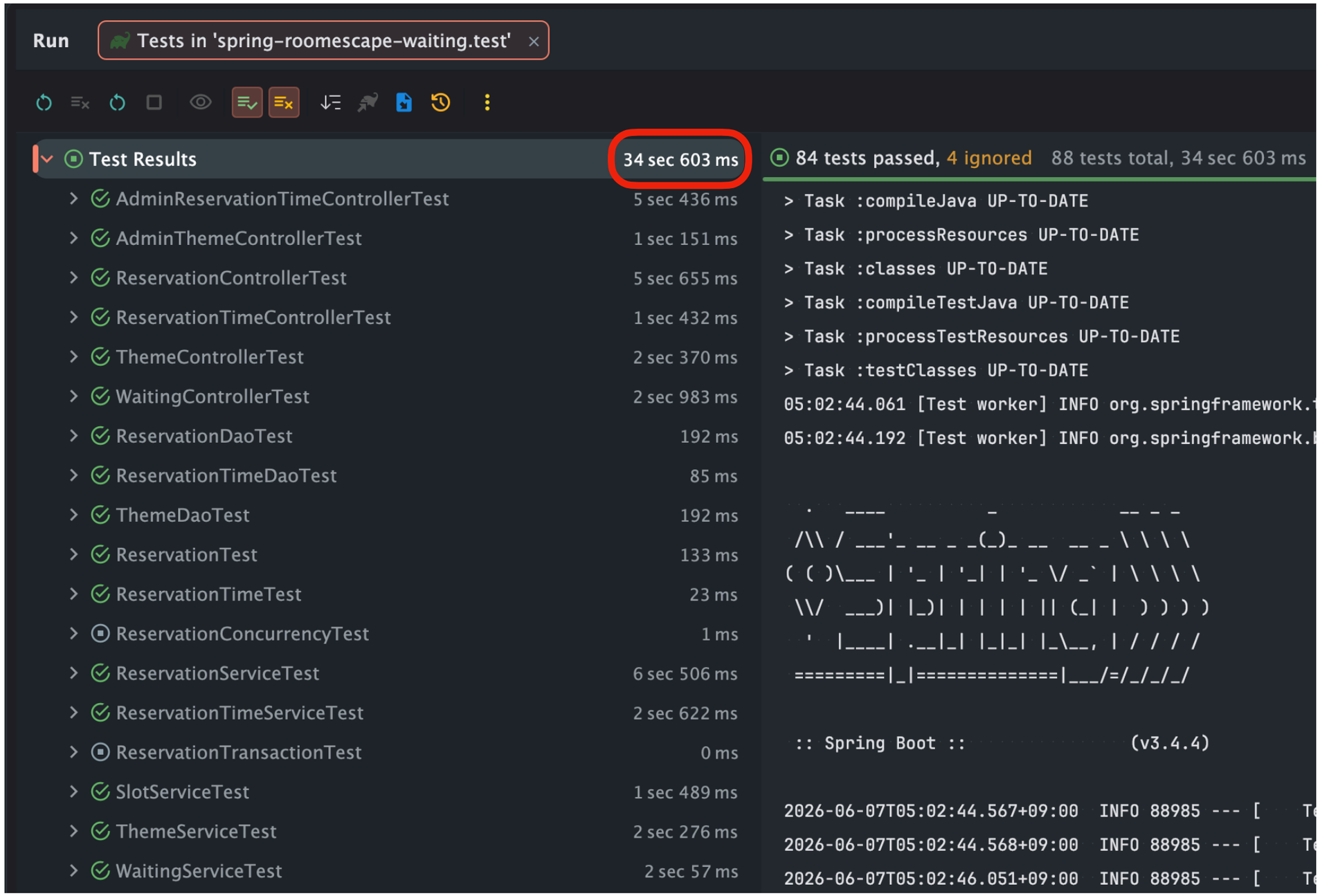
Spring ApplicationContext 는 왜 테스트 성능에 영향을 줄까?

발표자: 러키

문제 상황

테스트 속도가 너무 느리다!!

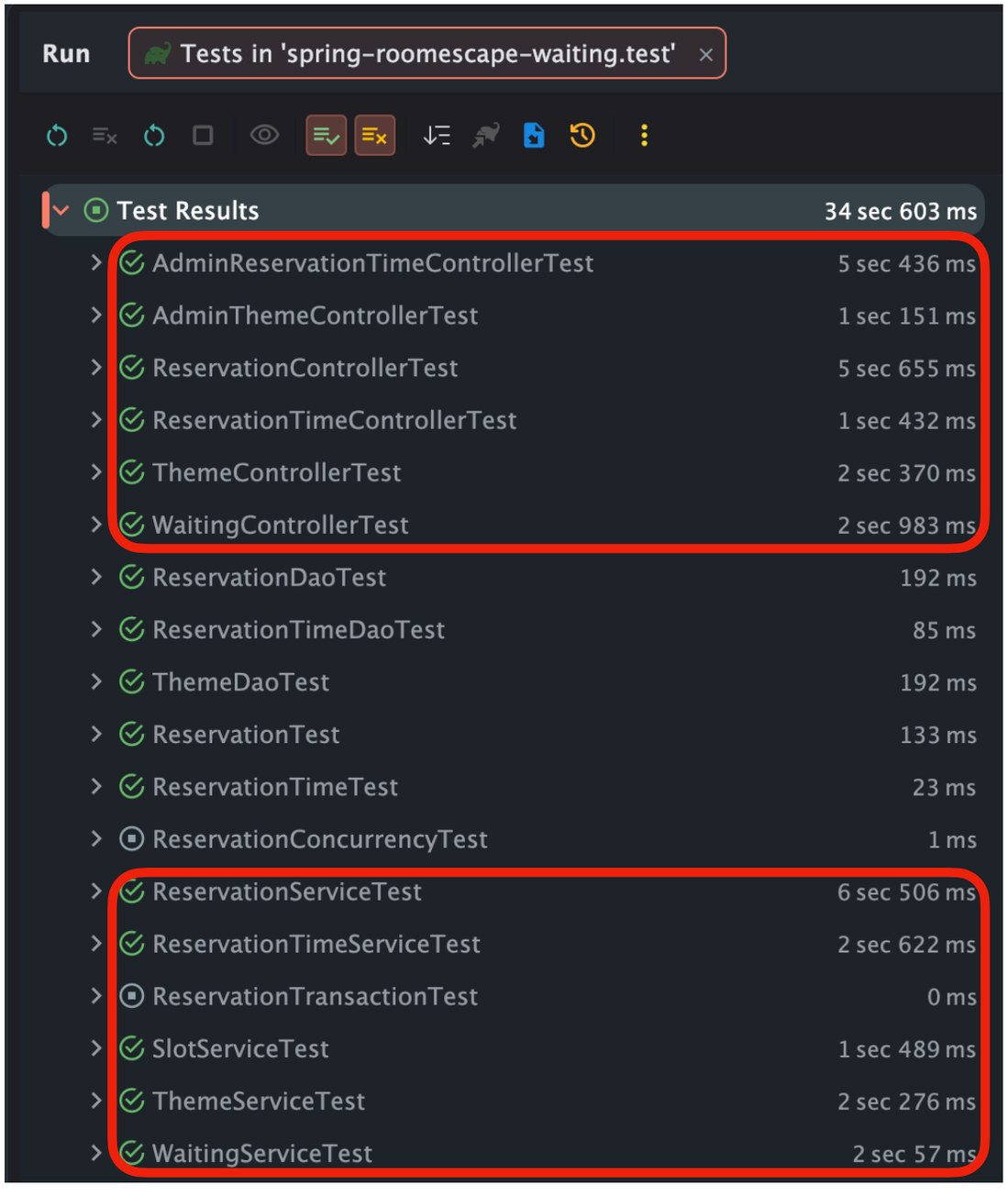


🛠️ 테스트 구조

계층	어노테이션	데이터 초기화 방식
Domain	없음 (순수 단위 테스트)	불필요
Repository	@JdbcTest	@Transactional 자동 롤백 (테스트 후)
Service	@SpringBootTest(None) + @DirtiesContext(BEFORE_EACH_TEST_METHOD)	컨텍스트 종료 -> H2 in-memory DB 드롭 -> 새 컨텍스트 -> scheme.sql 재실행
Controller	@SpringBootTest(DEFINED_PORT) + @DirtiesContext(BEFORE_EACH_TEST_METHOD)	컨텍스트 종료 -> H2 in-memory DB 드롭 -> 새 컨텍스트 -> scheme.sql 재실행

코드 바로가기: https://github.com/Jiihyun/spring-roomescape-waiting/tree/a-lot-of-context/src/test

!! 문제 실마리 발견



→

```
@SpringBootTest(webEnvironment = WebEnvironment.DEFINED_PORT)
@DirtiesContext(classMode = ClassMode.BEFORE_EACH_TEST_METHOD)
@Import(FixtureGeneratorConfig.class)
```

@DirtiesContext 사용하는 공통점 발견!!

→

```
@SpringBootTest(webEnvironment = WebEnvironment.NONE)
@DirtiesContext(classMode = ClassMode.BEFORE_EACH_TEST_METHOD)
@Import(FixtureGeneratorConfig.class)
```

💬 @DirtiesContext란?

`@DirtiesContext` indicates that the underlying Spring `ApplicationContext` has been dirtied during the execution of a test (that is, the test modified or corrupted it in some manner—for example, by changing the state of a singleton bean) and should be closed. When an application context is marked as dirty, it is removed from the testing framework's cache and closed. As a consequence, the underlying Spring container is rebuilt for any subsequent test that requires a context with the same configuration metadata.

이는 테스트 실행 중에 기본 Spring ApplicationContext가 더러워졌음을 나타내며(즉, 테스트가 어떤 식으로든 수정하거나 손상시켰음을 나타냅니다 — 예를 들어 싱글턴 빈의 상태를 변경하는 등) 닫아야 합니다. 애플리케이션 컨텍스트가 더티로 표시되면, 테스트 프레임워크의 캐시에서 제거되고 종료됩니다. 결과적으로, 동일한 구성 메타데이터를 가진 컨텍스트가 필요한 후속 테스트를 위해 기본 Spring 컨테이너가 재구축됩니다.

출처: https://docs.spring.io/spring-framework/reference/testing/annotations/integration-spring/annotation-dirtiescontext.html#page-title

- **@DirtiesContext(classMode = BEFORE_EACH_TEST_METHOD)**

  : 매 테스트마다 ApplicationContext를 새로 띄우게 만드는 설정

🎯 문제 원인

■ Controller

클래스	테스트 수	컨텍스트 수
AdminReservationTimeControllerTest	2	2
AdminThemeControllerTest	2	2
ReservationControllerTest	6	6
ReservationTimeControllerTest	1	1
ThemeControllerTest	2	2
WaitingControllerTest	3	3
소계	**16**	**16**

■ Service

클래스	테스트 수	컨텍스트 수
ReservationServiceTest	12	12
ReservationTimeServiceTest	7	7
ThemeServiceTest	6	6
WaitingServiceTest	5	5
SlotServiceTest	3	3
소계	**33**	**33**

■ Repository

클래스	컨텍스트 수
3개 클래스 전부 동일 `@Import` 구성	**1 (공유)**

생성된 컨텍스트: 총 50개

ApplicationContext란 무엇인가

Central interface to provide configuration for an application.

-> 애플리케이션 구성을 제공하는 중앙 인터페이스

```java
public interface ApplicationContext extends EnvironmentCapable, ListableBeanFactory, HierarchicalBeanFactory,
        MessageSource, ApplicationEventPublisher, ResourcePatternResolver {


    @Nullable
    String getId();

    String getApplicationName();

    String getDisplayName();

    long getStartupDate();

    @Nullable
    ApplicationContext getParent();

    AutowireCapableBeanFactory getAutowireCapableBeanFactory() throws IllegalStateException;

}
```

■ 역할

- Bean 접근 기능
- 리소스 로딩 기능
- 이벤트 발행 기능
- 국제화 기능 등

ApplicationContext가 뜰 때 하는 일

1. Classpath 스캔 & BeanDefinition 등록

JVM

scan

@RestController
@Service
@Repository
@RestController Advice
@Configuration

@Component 계열 클래스 발견

beanName	reservation Controller
beanClassName	roomescape.controller.ReservationController
constructorArg Values	[0] type = ReservationService
autowireCandidate	TRUE
scope	singleton

BeanDefinition 생성

{"reservationController, BeanDefinition"}

BeanDefinitionRegistry 에 Map으로 저장

ApplicationContext가 뜰 때 하는 일

2. Bean 생성 및 의존성 주입

DefaultListableBeanFactory.getBean(reservationController) 호출

↓

getBean(reservationService) 호출

```
public class ReservationService {  Complexity is 4 Everything is

    private final SlotService slotService;

    private final ReservationDao reservationDao;  10 usage
    private final ReservationTimeDao reservationTimeDao;
    private final ThemeDao themeDao;  2 usage
    private final WaitingDao waitingDao;
```

↓

getBean(slotService) 호출

↓

getBean(slotDao) 호출

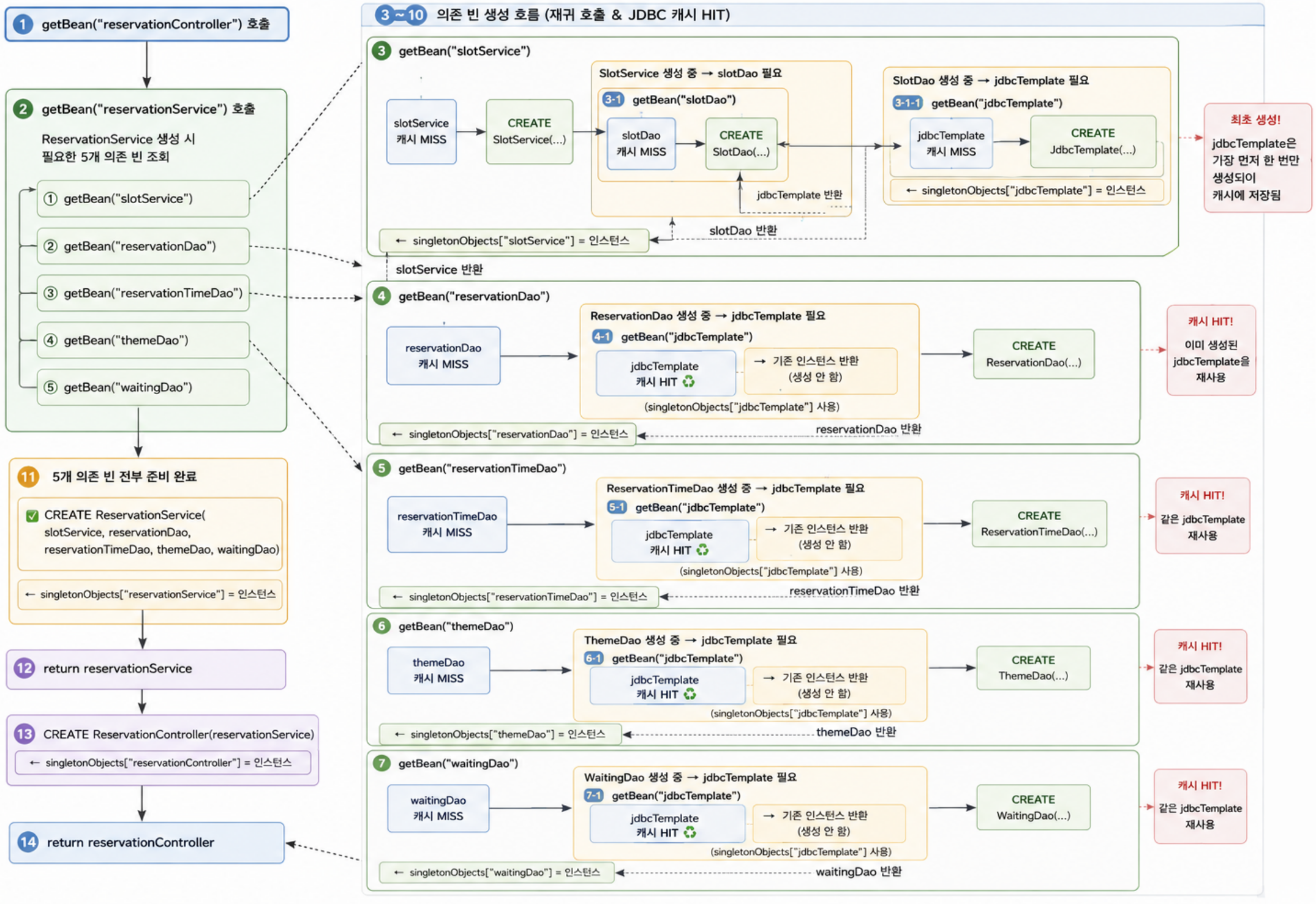
↓

getBean(jdbcTemplate) 호출

jdbcTemplate는 가장 먼저 딱 한 번만 생성 되며, 이후 중복 생성 없이 재사용 한다

ApplicationContext가 뜰 때 하는 일

2. Bean 생성 및 의존성 주입



ApplicationContext가 뜰 때 하는 일

3. BeanPostProcessor 실행

InfrastructureAdvisorAutoProxyCreator
.postProcessAfterInitialization(reservationService, "reservationService")

이 빈에 적용해야 할 @Transactional, @Async, @Cacheable 등이 있는지 검사

@Transactional 존재 ⭕

ReservationService$$SpringCGLIB$$0(프록시) 생성
&
프록시를 빈으로 등록

@Transactional 존재 ❌

원본 그대로 반환

🏭 ApplicationContext는 왜 비싼가?

모든 클래스 탐색 → 객체 생성 → 의존성 주입
→ BeanPostProcessor 실행 → 프록시 생성
과정을 모든 Bean에 대해 수행

🏭 ApplicationContext는 왜 비싼가?

회사 전체 조직도를 만들고 각 부서를 연결한 뒤
운영 준비까지 마치는 수준

-> 수백~수천 개

객체의 관계를 구성하는 비용 때문에 비싼 것

개선 아이디어

컨텍스트 생성을 최소화 하자!

with Context Caching

Context Caching이란?

Once the TestContext framework loads an ApplicationContext (or WebApplicationContext) for a test, that context is cached and reused for all subsequent tests that declare the same unique context configuration within the same test suite.

출처: https://docs.spring.io/spring-framework/reference/testing/testcontext-framework/ctx-management/caching.html

-> TestContext 프레임워크가 테스트용 ApplicationContext(또는 WebApplicationContext)를 로드하면,
그 컨텍스트는 캐시되어 동일한 테스트 스위트 내에서
동일한 고유 컨텍스트 구성을 선언하는 모든 후속 테스트에 재사용

Context Caching 조건

- 동일한 설정 클래스 (@SpringBootTest, @ContextConfiguration)

- 동일한 Active Profile (@ActiveProfiles)

- 동일한 Property 설정 (@SpringBootTest(properties), @TestPropertySource)

- 동일한 MockBean / SpyBean 구성

- @DirtiesContext 미사용

조건(Key)이 하나라도 달라지면 새로운 ApplicationContext 생성!

Context Caching 조건

class 1

@MockBean UserRepository

class 2

@MockBean UserRepository

@MockBean MailSender

@MockBean UserRepository

@MockBean PaymentRepository

class 3

@MockBean UserRepository

@MockBean PaymentRepository

@MockBean MailSender

class 4

Context Caching 조건

class 1

@MockBean UserRepository

class 2

@MockBean UserRepository

@MockBean MailSender

class 3

@MockBean UserRepository

@MockBean PaymentRepository

class 4

@MockBean UserRepository

@MockBean PaymentRepository

@MockBean MailSender

Context 4개 생성

개선 방식 1

@DirtiesContext 제거
&
테스트 데이터 초기화 클래스 도입

```java
@Component  2 usages  Jihyun Lim
public class DatabaseCleaner {

    private final JdbcTemplate jdbcTemplate;  6 usages

    public DatabaseCleaner(JdbcTemplate jdbcTemplate) {  Jihyun Lim
        this.jdbcTemplate = jdbcTemplate;
    }

    public void clean() {  2 usages  Jihyun Lim
        jdbcTemplate.update( sql: "DELETE FROM waiting");  SQL dialect
        jdbcTemplate.update( sql: "DELETE FROM reservation");  SQL di
        jdbcTemplate.update( sql: "DELETE FROM slot");  SQL dialect is
        jdbcTemplate.update( sql: "DELETE FROM reservation_time");  S
        jdbcTemplate.update( sql: "DELETE FROM theme");  SQL dialect i
    }
}
```

개선 방식 2

계층별 컨텍스트 종류 통합

```
@Import(FixtureGeneratorConfig.class)  12 usages  6 inheritors  Jihyun Lim
@SpringBootTest(webEnvironment = SpringBootTest.WebEnvironment.RANDOM_PORT)
public abstract class AcceptanceTest {

    @LocalServerPort
    private int port;

    @Autowired
    private DatabaseCleaner databaseCleaner;

    @Autowired
    protected ApiFixtureGenerator apiFixtureGenerator;

    @BeforeEach  Jihyun Lim
    void setUp() {
        RestAssured.port = port;
        databaseCleaner.clean();
    }
}
```

```
@Import(FixtureGeneratorConfig.class)  14 usages  7 inheritors  Jihyun Lim
@SpringBootTest(webEnvironment = SpringBootTest.WebEnvironment.NONE)
public abstract class ServiceTest {

    @Autowired
    private DatabaseCleaner databaseCleaner;

    @Autowired
    protected FixtureGenerator fixtureGenerator;

    @BeforeEach  Jihyun Lim
    void setUp() {
        databaseCleaner.clean();
    }
}
```

적용 결과

■ Controller

클래스	테스트 수	컨텍스트
AdminReservationTimeControllerTest	2	① @SpringBootTest(RANDOM_PORT)
AdminThemeControllerTest	2	①
ReservationControllerTest	6	①
ReservationTimeControllerTest	1	①
ThemeControllerTest	2	①
WaitingControllerTest	3	①
소 계	16	1개

■ Service

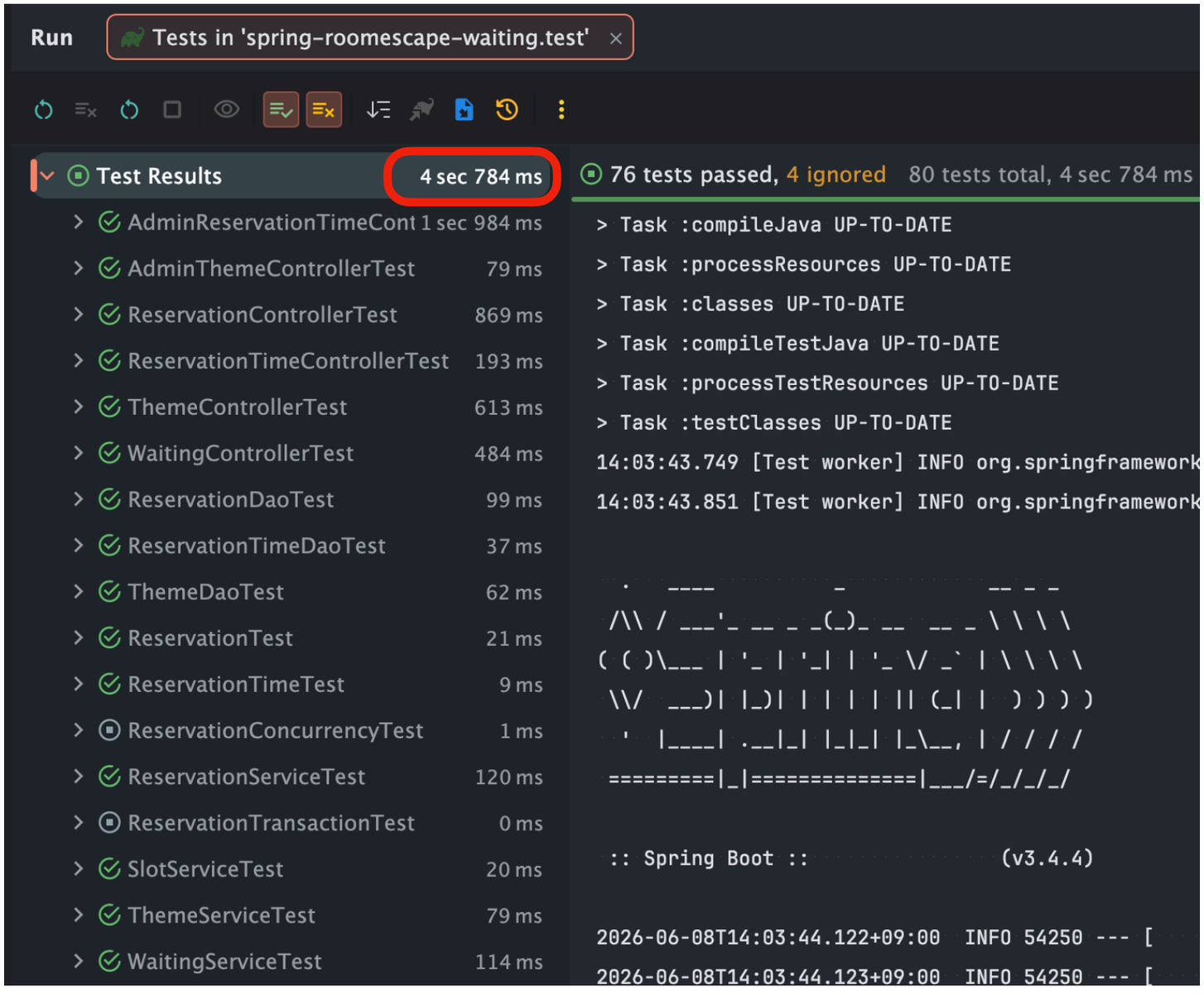
클래스	테스트 수	컨텍스트
ReservationServiceTest	12	② @SpringBootTest(NONE)
ReservationTimeServiceTest	7	②
SlotServiceTest	3	②
ThemeServiceTest	6	②
WaitingServiceTest	5	②
소 계	33	1개

■ Repository

클래스	테스트 수	컨텍스트
ReservationDaoTest	8	⑤ @JdbcTest
ReservationTimeDaoTest	6	⑤
ThemeDaoTest	4	⑤
소 계	18	1개

생성된 컨텍스트: 50개 -> 3개로 축소

적용 결과



실행 시간: 34s -> 4s 로 감소

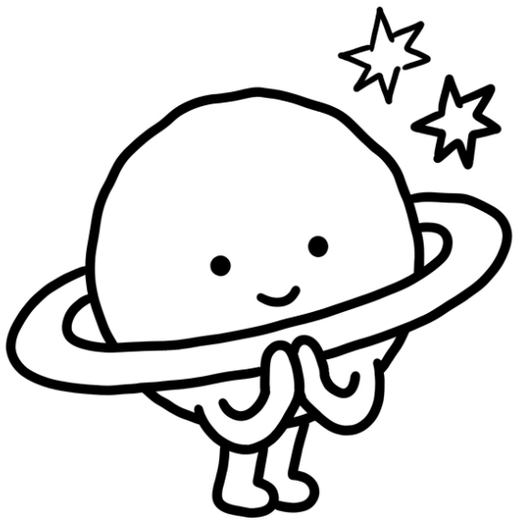
결론

- Spring 테스트의 성능은
ApplicationContext 에 영향을 받는다.

- ApplicationContext 는
Spring 서버를 부팅하는 과정이므로 비용이 크다.

- Spring은 이를 해결하기 위해 Context 캐싱을 제공한다.

- Context 캐시 히트율을 높여 속도를 개선시켜보자!



고