Spring Boot Caching - Detailed Notes

1. What is Caching?

Caching stores frequently accessed data in fast memory so that future requests are served quickly without expensive operations.

2. Why Use Caching?

Without Caching: Every request goes to the database -> Slower.

With Caching: First request hits DB, subsequent requests use cached result -> Faster and less load.

3. Spring Boot Caching Support

Spring provides annotations like @Cacheable, @CachePut, and @CacheEvict. Spring Boot supports pluggable cache providers like ConcurrentMap, Redis, EhCache, and Caffeine.

4. Basic Setup

- Add dependency: spring-boot-starter-cache
- Use @EnableCaching in main class

5. Core Annotations

@Cacheable - Caches the result based on method params.

@CachePut - Forces method execution and updates the cache.

@CacheEvict - Removes one or all entries from the cache.

6. Default Cache Manager

ConcurrentMapCacheManager (in-memory, not distributed).

Can be replaced with Redis, EhCache, or Caffeine for advanced usage.

7. Cache Key Customization

Use SpEL in `key` attribute for custom cache keys.

8. Conditions and Unless

Use `condition` to apply caching selectively.

Use `unless` to skip caching based on result.

9. Redis Integration (Advanced)

- Add spring-boot-starter-data-redis dependency.
- Configure Redis host and port.
- Use spring.cache.type=redis

10. Common Pitfalls

- Missing @EnableCaching
- Incorrect cache key
- Data not updating -> use @CachePut
- Cache size not managed -> use TTL

11. Summary Table (Expanded Explanation)

Annotation	Purpose
@Cacheable	Checks if data is in cache; if yes, returns cached data. If not, method runs and result is cached.
@CachePut	Always executes method and updates the cache with returned result. Useful for update operations.
@CacheEvict	Removes one or multiple cache entries. Essential when deleting or invalidating outdated cache data.

12. Sample Use Case:

```
@Cacheable(value = "employeeCache", key = "#id")
public Employee getEmpById(Long id) {
    return empRepository.findById(id).orElse(null);
}
```

13. Interview Questions:

1. What is caching and why is it important in microservices?
2. Explain the difference between @Cacheable and @CachePut.
3. What are the limitations of default caching in Spring Boot?
4. How would you use Redis with Spring Boot caching?
5. What are some common issues you might face with caching?

6. How do you manage cache invalidation in Spring?