**Basic Spring Boot Caching Questions**

**1. What is caching in Spring Boot?**

**Answer:**  
Caching is a mechanism to store frequently accessed data in memory (or a fast data store like Redis) to reduce computation time, database hits, and improve performance. Spring Boot provides caching support via @EnableCaching, @Cacheable, @CachePut, and @CacheEvict.

**2. How do you enable caching in Spring Boot?**

**Answer:**

java

CopyEdit

@Configuration

@EnableCaching

public class CacheConfig {

}

You also need a caching implementation like EhCache, Redis, or Caffeine configured.

**3. What is the use of @Cacheable annotation?**

**Answer:**  
It marks a method so that the result is stored in the cache. If the same method is called with the same parameters, the cached value is returned.

java

CopyEdit

@Cacheable(value = "products", key = "#id")

public Product getProductById(String id) {

// expensive DB call

}

**4. What is @CachePut used for?**

**Answer:**  
@CachePut updates the cache without interfering with the method execution.

java

CopyEdit

@CachePut(value = "products", key = "#product.id")

public Product updateProduct(Product product) {

return repository.save(product);

}

**5. What is @CacheEvict used for?**

**Answer:**  
Removes data from the cache. Example:

java

CopyEdit

@CacheEvict(value = "products", key = "#id")

public void deleteProduct(String id) {

repository.deleteById(id);

}

**🔹 Redis-Specific Questions**

**6. Why use Redis with Spring Boot?**

**Answer:**

* High-performance in-memory data store.
* Supports TTL (Time-to-Live).
* Persistent and scalable.
* Pub/Sub support for event-driven systems.

**7. How do you configure Redis in Spring Boot?**

**Answer:**  
In application.properties:

properties

CopyEdit

spring.cache.type=redis

spring.redis.host=localhost

spring.redis.port=6379

Add dependency:

xml

CopyEdit

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-data-redis</artifactId>

</dependency>

**8. How is TTL managed in Redis caching in Spring?**

**Answer:**  
Use RedisCacheConfiguration to set TTL:

java

CopyEdit

@Bean

public RedisCacheManager cacheManager(RedisConnectionFactory factory) {

RedisCacheConfiguration config = RedisCacheConfiguration.defaultCacheConfig()

.entryTtl(Duration.ofMinutes(10));

return RedisCacheManager.builder(factory).cacheDefaults(config).build();

}

**9. What serialization strategies are used with Redis?**

**Answer:**  
Default is Java serialization (JDK serialization). You can customize with:

* Jackson JSON (GenericJackson2JsonRedisSerializer)
* Kryo
* String RedisSerializer

Example:

java

CopyEdit

template.setValueSerializer(new GenericJackson2JsonRedisSerializer());

**10. What are common Redis cache issues in Spring Boot?**

**Answer:**

* Serialization/deserialization failures.
* Cache stampede or penetration.
* Cache consistency with DB.
* Connection pool exhaustion.

**🔹 Advanced & Cross Questions**

**11. How would you handle cache eviction on DB updates?**

**Answer:**  
Use @CacheEvict after @Transactional DB operation to remove stale data from the cache. You can also use Redis Pub/Sub to broadcast cache invalidation messages.

**12. Can you have multiple cache managers in Spring Boot?**

**Answer:**  
Yes, configure custom cache managers using @Primary and qualifiers.

**13. What is cache penetration and how to avoid it in Redis?**

**Answer:**  
It occurs when requests query keys that don't exist in cache or DB, overwhelming DB.  
**Solution:**

* Cache null responses.
* Use Bloom filters to avoid unnecessary lookups.

**14. What is cache stampede and how do you prevent it?**

**Answer:**  
Multiple threads query the cache simultaneously for a missing key and hit DB together.

**Solution:**

* Locking (Mutex)
* Prewarm caches
* Randomized TTL

**15. How do you cache method results conditionally in Spring Boot?**

**Answer:**  
Use condition and unless attributes:

java

CopyEdit

@Cacheable(value = "users", key = "#id", unless = "#result == null")

public User findUserById(String id) {}

**🔹 Coding/Design Task Question Example**

**Design a Spring Boot service to cache user profile data in Redis with a TTL of 5 minutes, and allow updating and eviction.**

**Answer Summary:**

* Use @Cacheable, @CachePut, and @CacheEvict.
* Configure TTL in RedisCacheManager.
* Use Jackson for serialization.
* Ensure DB consistency with cache.

**🔹 Tools & Libraries**

* **Spring Cache Abstraction**
* **Redis (Lettuce / Jedis)**
* **Caffeine**
* **EhCache**
* **Hazelcast**
* **Micrometer** (for cache metrics)

**🔹 Popular Java Caching Libraries and Frameworks**

| **Library / Framework** | **Type** | **In-Memory / Distributed** | **Notes** |
| --- | --- | --- | --- |
| **Spring Cache Abstraction** | API layer | Wrapper | Needs backing implementation |
| **EhCache** | Java-based | In-Memory / Disk | Simple, popular, easy to integrate |
| **Caffeine** | Java-based | In-Memory | Fastest in-memory cache |
| **Guava Cache** | Java-based | In-Memory | Lightweight, part of Google libraries |
| **Redis** | External (remote) | Distributed (in-memory) | Persistent, scalable |
| **Hazelcast** | Distributed | In-Memory + Network | Clustered, P2P, supports JCache |
| **Infinispan** | Distributed | In-Memory + Disk | Red Hat, highly scalable |
| **Memcached** | External (remote) | Distributed | Simple, no persistence |
| **Apache Ignite** | Distributed | In-Memory + Compute Grid | For caching + computation |

**🔹 Quick Comparison Table**

| **Feature** | **EhCache** | **Caffeine** | **Redis** | **Hazelcast** | **Infinispan** |
| --- | --- | --- | --- | --- | --- |
| Speed | ⚡⚡ | ⚡⚡⚡⚡⚡ | ⚡⚡⚡ | ⚡⚡⚡ | ⚡⚡⚡ |
| TTL Support | ✅ | ✅ | ✅ | ✅ | ✅ |
| Persistence | ✅ (Disk) | ❌ | ✅ | Partial | ✅ |
| Cluster Support | ❌ (Eh2 yes) | ❌ | ✅ | ✅ | ✅ |
| Spring Boot Integration | ✅ | ✅ | ✅ | ✅ | ✅ |
| JCache Support | ✅ | ❌ | ❌ | ✅ | ✅ |
| Best Use Case | Local cache | Hot path LRU | Shared cache | Cluster + P2P | Distributed compute/cache |

**🔹 When to Use Which Cache**

| **Use Case** | **Best Option** |
| --- | --- |
| **Simple local caching** | Caffeine or EhCache |
| **High-performance in-memory cache** | Caffeine |
| **Shared cache across instances** | Redis |
| **Clustered microservices** | Hazelcast or Infinispan |
| **Large-scale, compute + cache** | Apache Ignite |
| **Lightweight temporary cache** | Guava Cache |

**🔹 Best Caching Library?**

**✅ Local In-Memory Best: Caffeine**

* Fastest.
* Good for hot-path data (rate-limiting, feature flags, etc.)
* Used by LinkedIn, Spotify.

**✅ Distributed Best: Redis**

* Popular with Spring Boot.
* Scalable, supports eviction, persistence, TTL.
* Good for microservices and APIs.

**✅ Clustered Enterprise: Hazelcast or Infinispan**

* Used in high availability environments.
* Peer-to-peer data replication.

**🔹 Spring Boot Integration Example**

**1. Using Caffeine**

xml

CopyEdit

<dependency>

<groupId>com.github.ben-manes.caffeine</groupId>

<artifactId>caffeine</artifactId>

</dependency>

yaml

CopyEdit

spring.cache.type=caffeine

spring.cache.caffeine.spec=maximumSize=500,expireAfterWrite=10m

**2. Using Redis**

xml

CopyEdit

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-data-redis</artifactId>

</dependency>

yaml

CopyEdit

spring.cache.type=redis

spring.redis.host=localhost

spring.redis.port=6379

**🔹 Final Recommendation (Summary):**

| **Scenario** | **Recommendation** |
| --- | --- |
| Fast local caching | ✅ Caffeine |
| Cache per service / no network latency | ✅ EhCache |
| Multiple instances (shared cache) | ✅ Redis |
| Distributed cache in cluster | ✅ Hazelcast / Infinispan |
| Analytics + caching | ✅ Apache Ignite |