To Be Implemented

```
public interface CacheService {
    void put(String key, Object value);
    Object get(String key);
    void evict(String key);
}
```

2️⃣ Create a dynamic Redis implementation

java

Copy code

```
import org.springframework.data.redis.connection.lettuce.LettuceConnectionFactory;
import org.springframework.data.redis.core.StringRedisTemplate;
import org.springframework.stereotype.Service;

@Service("dynamicRedisCache")
public class DynamicRedisCacheService implements CacheService {

    private StringRedisTemplate redisTemplate;

    public void setRedisConnection(String host, int port, String password) {
        LettuceConnectionFactory factory = new LettuceConnectionFactory(host, port);
        if (password != null && !password.isEmpty()) {
            factory.setPassword(password);
        }
        factory.afterPropertiesSet();
        this.redisTemplate = new StringRedisTemplate(factory);
    }

    @Override
    public void put(String key, Object value) {
        redisTemplate.opsForValue().set(key, value.toString());
    }

    @Override
    public Object get(String key) {
        return redisTemplate.opsForValue().get(key);
    }

    @Override
    public void evict(String key) {
        redisTemplate.delete(key);
    }
}
```

✅ Key idea: setRedisConnection allows you to create a new connection per request.

3️⃣ Factory or Service to choose cache

java

Copy code

```
import org.springframework.stereotype.Component;

@Component
public class CacheFactory {

    private final DynamicRedisCacheService redisCache;
    // Add other caches if needed (Elasticache, etc.)

    public CacheFactory(DynamicRedisCacheService redisCache) {
        this.redisCache = redisCache;
    }

    public CacheService getCache(String cacheType, Map<String, String> params) {
        switch (cacheType.toLowerCase()) {
            case "redis":
                String host = params.get("host");
                int port = Integer.parseInt(params.getOrDefault("port", "6379"));
                String password = params.get("password");
                redisCache.setRedisConnection(host, port, password);
                return redisCache;
            // case "elastic": return elasticCache;
            default:
                throw new IllegalArgumentException("Unknown cache type: " + cacheType);
        }
    }
}
```

4️⃣ Controller usage

java

Copy code

```
@RestController
@RequestMapping("/cache")
public class CacheController {

    private final CacheFactory cacheFactory;

    public CacheController(CacheFactory cacheFactory) {
        this.cacheFactory = cacheFactory;
    }

    @PostMapping("/set")
    public String setValue(@RequestParam String cacheType,
                           @RequestParam String key,
                           @RequestParam String value,
                           @RequestParam(required = false) String host,
                           @RequestParam(required = false) Integer port,
                           @RequestParam(required = false) String password) {

        Map<String, String> params = Map.of(
            "host", host != null ? host : "127.0.0.1",
            "port", port != null ? port.toString() : "6379",
            "password", password != null ? password : ""
        );

        CacheService cache = cacheFactory.getCache(cacheType, params);
        cache.put(key, value);
        return "OK";
    }

    @GetMapping("/get")
    public Object getValue(@RequestParam String cacheType,
                           @RequestParam String key,
                           @RequestParam(required = false) String host,
                           @RequestParam(required = false) Integer port,
                           @RequestParam(required = false) String password) {

        Map<String, String> params = Map.of(
            "host", host != null ? host : "127.0.0.1",
            "port", port != null ? port.toString() : "6379",
            "password", password != null ? password : ""
        );

        CacheService cache = cacheFactory.getCache(cacheType, params);
        return cache.get(key);
    }
}
```

✅ Advantages of this pattern

Dynamic cache selection: Each request decides which cache to use.

Dynamic Redis connections: Supports multi-tenant Redis setups or different Redis instances per request.

Extensible: Just add another case in CacheFactory for Elasticache, Memcached, etc.

Single interface: Your service/controller code doesn’t care which cache is used.

If you want, I can also refactor this further using Spring’s @Configuration and

BeanFactory so that each Redis connection is a real Spring-managed bean instead of recreating StringRedisTemplate every request. This is cleaner for production systems.

Do you want me to show that version too?

You said:
i liked this but iam not using rest as i want fast iam using GRPC

ChatGPT said:
Ah, perfect — that actually fits really nicely! gRPC just changes the transport layer; the core caching pattern doesn't change at all. You just call your dynamic cache service from the gRPC service implementation instead of a REST controller.

Here's how to adapt the previous dynamic cache pattern for gRPC:

1️⃣ Keep the same CacheService and DynamicRedisCacheService
No changes here — your CacheService interface and DynamicRedisCacheService remain exactly the same.

2️⃣ gRPC service definition (proto)
Example cache.proto:

proto
Copy code

```
syntax = "proto3";

package cache;

service CacheService {
  rpc SetValue(SetRequest) returns (SetResponse);
  rpc GetValue(GetRequest) returns (GetResponse);
}

message SetRequest {
  string cache_type = 1;
  string key = 2;
  string value = 3;
  string host = 4;
  int32 port = 5;
  string password = 6;
}
```

```
message SetResponse {
    string status = 1;
}

message GetRequest {
    string cache_type = 1;
    string key = 2;
    string host = 3;
    int32 port = 4;
    string password = 5;
}

message GetResponse {
    string value = 1;
}
```

3️⃣ Implement the gRPC service in Spring Boot

java

Copy code

```
import io.grpc.stub.StreamObserver;
import net.devh.boot.grpc.server.service.GrpcService;

import java.util.Map;

@GrpcService
public class CacheGrpcService extends
CacheServiceGrpc.CacheServiceImplBase {

    private final CacheFactory cacheFactory;

    public CacheGrpcService(CacheFactory cacheFactory) {
        this.cacheFactory = cacheFactory;
    }

    @Override
    public void setValue(SetRequest request, StreamObserver<SetResponse>
responseObserver) {
        Map<String, String> params = Map.of(
            "host", request.getHost().isEmpty() ? "127.0.0.1" : request.getHost(),
            "port", request.getPort() == 0 ? "6379" :
String.valueOf(request.getPort()),
            "password", request.getPassword()
        );

        CacheService cache = cacheFactory.getCache(request.getCacheType(),
params);
        cache.put(request.getKey(), request.getValue());
        SetResponse response = SetResponse.newBuilder().setStatus("OK").build();
        responseObserver.onNext(response);
        responseObserver.onCompleted();
    }

    @Override
    public void getValue(GetRequest request, StreamObserver<GetResponse> responseObserver) {
        Map<String, String> params = Map.of(
            "host", request.getHost().isEmpty() ? "127.0.0.1" : request.getHost(),
            "port", request.getPort() == 0 ? "6379" : String.valueOf(request.getPort()),
            "password", request.getPassword()
        );

        CacheService cache = cacheFactory.getCache(request.getCacheType(), params);
        Object value = cache.get(request.getKey());

        GetResponse response = GetResponse.newBuilder()
                .setValue(value != null ? value.toString() : "")
                .build();
        responseObserver.onNext(response);
        responseObserver.onCompleted();
    }
}
```