短域名服务

需求背景

> 撰写两个 API 接口:
>
> - 短域名存储接口：接受长域名信息，返回短域名信息
> - 短域名读取接口：接受短域名信息，返回长域名信息。
>
> 限制:
>
> - 短域名长度最大为 8 个字符
> - 采用SpringBoot，集成Swagger API文档；

> - JUnit编写单元测试，使用Jacoco生成测试报告(测试报告提交截图)；
> - 映射数据存储在JVM内存即可，防止内存溢出；

需求分析

功能拆解

系统主要有两个功能：i.长域名转换为短域名 ii.短域名转换为长域名，长域名转换为短域名可以使用算法实现，短域名转换为长域名需要先保存短长域名的映射关系，然后根据短域名获取长域名。

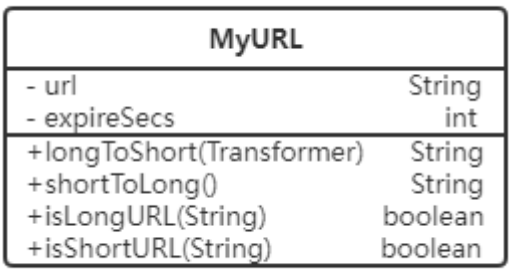
模块	功能点
url转换	i.实现转换算法对url进行转换。 ii.预留转换算法扩展接口。
url存储	i.对长短url链接映射进行持久化。 ii.用户可指定url生效时间。
质量保证	i.使用Junit编写单元测试。 ii.使用Jacoco生成测试报告。

领域建模

本需求中领域模型可抽象为url实体。

该实体的属性为url链接地址，实体方法为针对url链接进行转换。

为了保证扩展性，需要由上级将转换算法作为参数传入url实体中，类图如下



性能要求

长短域名解析接口流量较大，应该使用缓存来保证系统的可用性。

技术选型

1.框架：使用DDD架构，解耦业务流程，增强领域内聚，保证业务逻辑的可复用性和扩展性。

2.转换算法：

算法	原理	优点	缺点
自增序列算法	生成自增id（可用mysql主键id、雪花算法等），然后id转化为62个字符[a – z, A – Z, 0 – 9]，10进制转62进制，8个字符总共可以表示62^8≈218万亿。	不会重复。	需要增加第3方依赖复杂化，mysql主键id依赖mysql，雪花算法机器id分配依赖第3方存储，单机情况下也可硬编码，同时存在时钟回拨问题，最简单的解决方式是直接抛异常。
摘要算法	先md5生成32位加密串，取前10个字符，看成16进制对应40个bit位，每5个bit位可表示0~63，可对应到62个字符[a – z, A –Z, 0 – 9]的index，40个bit位共可表示8位这样的字符，合起来即为所需的8位短串。	内存操作，简单可靠。	可能有重复，概率极小。

本次需求选择摘要算法。

3.持久化：

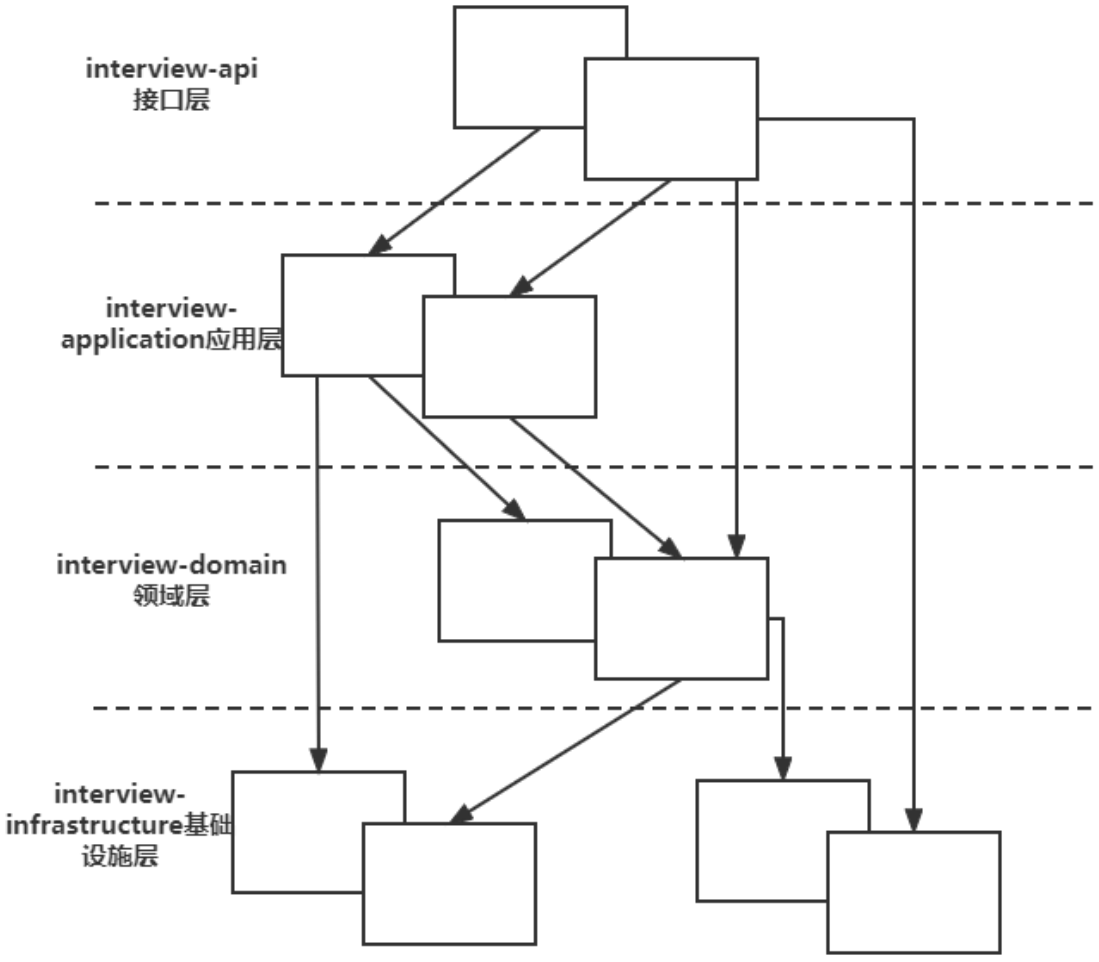
理想持久化架构为分布式缓存+本地缓存，高并发情况下可以通过本地缓存缓存热点数据减轻分布式缓存压力，设置自动加载，本地缓存未查询到则降级到分布式缓存。

分布式缓存：redis，可以通过过期时间有效解决用户指定短链接生效时间的问题，本次需求考虑成本问题暂不实现。

本地缓存：caffine，采用按容量淘汰与按时间淘汰双重淘汰策略来防止内存溢出。W-TinyLFU算法能有效保证缓存的命中率。

需求设计

架构设计

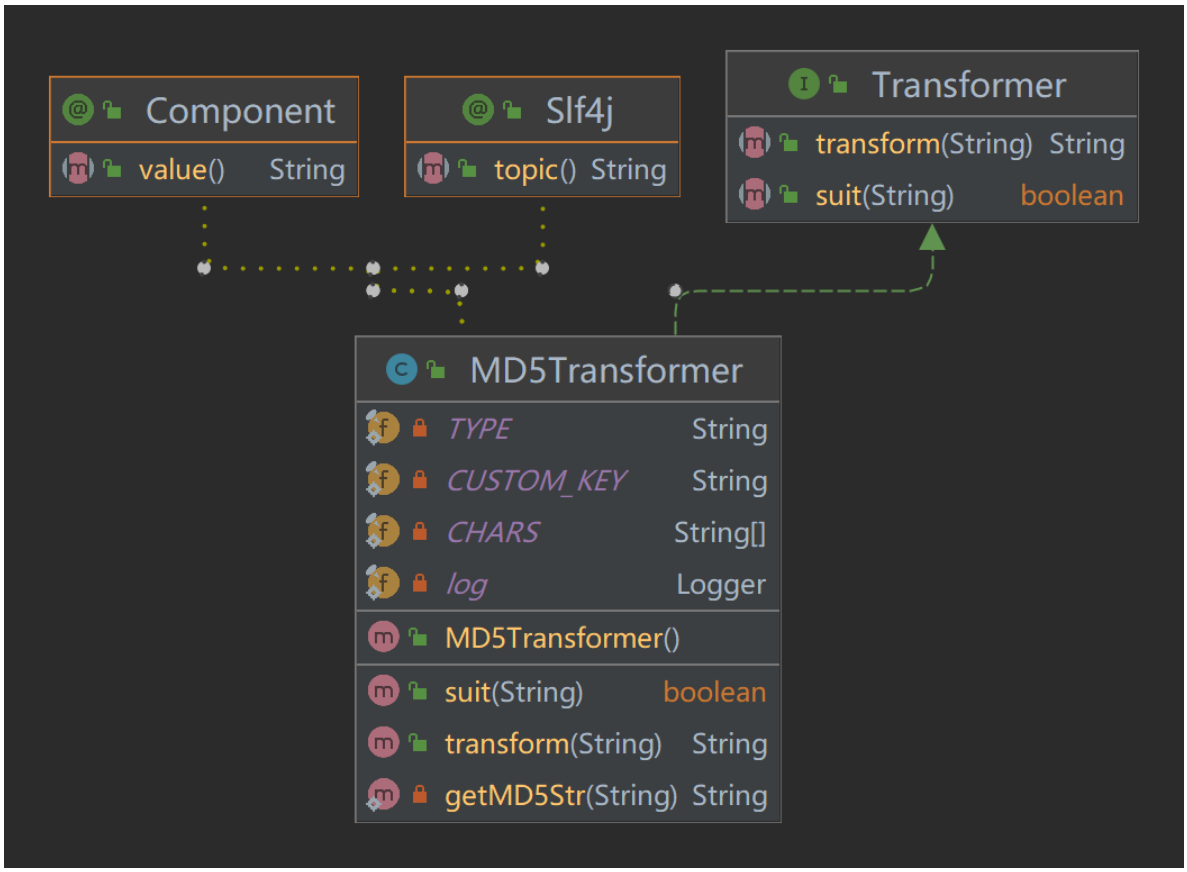


采用DDD架构，其中domain层位于最下层，infrastructure层依赖反转，repository包的接口位于domain层中。

模块设计

- interview-api：承载外部接口和接口相关的配置，该需求中用于存放controller、启动类和单测。
- interview-application：承载业务流程处理和对领域对象的编排，该需求中用于根据用户指定的算法组装转换算法与url实体。
- interview-domain：承载领域能力，该需求中用于存放领域模型、工厂类及一些基本能力。
- interview-infrastructure：承载底层技术的实现，该需求中用于实现本地缓存与转换算法。

转换算法类图如下：



算法的具体实现类实现Transformer接口，通过domain层的factory来生成转换算法对象。

未来期望

1.后续可接入redis来实现短链接系统的产品化。

2.后续可集成多种转换算法供用户进行选择。

3.后续可对服务做set化，按照区域进行路由，使得本地缓存能够根据不同的区域更个性化地缓存热点数据。

质量保证

测试方案

1.集成测试，通过mockHttpRequest来调用rest接口，验证接口功能是否正常。

```
public BaseResponse<TransformResult> mockHttpRequest(String url, Object request) {
        ObjectMapper objectMapper = new ObjectMapper();
        try {
            MvcResult mvcResult = mockMvc.perform(post(url)
                                .accept(MediaType.APPLICATION_JSON)
                                .characterEncoding("UTF-8")
                                .contentType(MediaType.APPLICATION_JSON)

.content(objectMapper.writeValueAsString(request)))
                    .andExpect(status().isOk())

.andExpect(content().contentType(MediaType.APPLICATION_JSON))
                    .andReturn();

            String json = mvcResult.getResponse().getContentAsString(Charsets.UTF_8);
            return objectMapper.readValue(json, new TypeReference<BaseResponse<TransformResult>>() {
            });
        } catch (Exception e) {
            log.error("mockHttpRequest error", e);
            return null;
        }
    }
```

2.单元测试，通过Junit和Mockito来mock程序的各种异常状态进行功能验证。

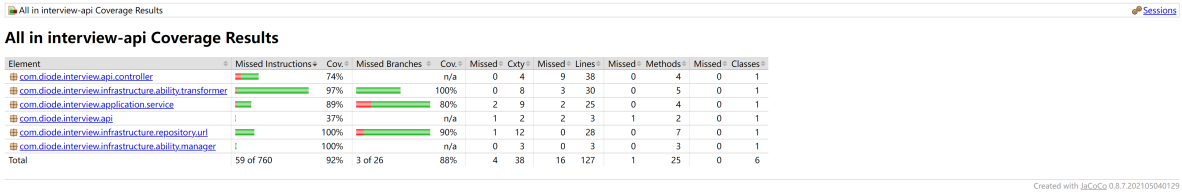
```
@Slf4j
@SpringBootTest
@RunWith(MockitoJUnitRunner.class)
```

```java
public class MyURLTest {

    @Mock
    private MD5Transformer transformer;

    @InjectMocks
    private MyURL myURL;


    @Test
    public void testLongToShort(){

        myURL.setUrl(null);
        String s = myURL.longToShort(transformer);
        Assert.assertTrue(Strings.isNullOrEmpty(s));

        myURL.setUrl("aaaaaa");
        Assertions.assertThrows(BizException.class, () ->
myURL.longToShort(transformer));

        String s2 = myURL.longToShort(null);
        Assert.assertTrue(Strings.isNullOrEmpty(s2));
    }
}
```

测试报告

100% classes, 87% lines covered in package 'com.diode'

Element	Class, %	Method, %	Line, %	Branch, %
interview	100% (6/6)	94% (18/19)	87% (111/...	88% (23/26)


All in interview-api Coverage Results

Sessions

All in interview-api Coverage Results

Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed	Cxty	Missed	Lines	Missed	Methods	Missed	Classes
com.diode.interview.api.controller		74%		n/a	0	4	9	38	0	4	0	1
com.diode.interview.infrastructure.ability.transformer		97%		100%	0	8	3	30	0	5	0	1
com.diode.interview.application.service		89%		80%	2	9	2	25	0	4	0	1
com.diode.interview.api		37%		n/a	1	2	2	3	1	2	0	1
com.diode.interview.infrastructure.repository.url		100%		90%	1	12	0	28	0	7	0	1
com.diode.interview.infrastructure.ability.manager		100%		n/a	0	3	0	3	0	3	0	1
Total	59 of 760	92%	3 of 26	88%	4	38	16	127	1	25	0	6

Created with JaCoCo 0.8.7.202105040129

性能测试

硬件环境

内存:16g

CPU:Intel(R) Core(TM) i7-1065G7 CPU @ 1.30GHz　1.50 GHz

运行环境

Jmeter

> Ramp-Up时间：3s
>
> 循环次数：10

Tomcat

> server.tomcat.max-threads = 1000
>
> server.tomcat.max-connections = 20000

JVM

> -Xms4028M
>
> -Xmx4028M
>
> -Xmn1024M
>
> -Xss1M

Caffeine

> maxSize:1000000

测试结果

Label	# 样本	平均值	最小值	最大值	标准偏差	异常 % ↓	吞吐量	接收 KB/sec	发送 KB/sec	平均字节数
长-短	231000	16	0	1246	25.46	0.00%	437.5/sec	95.58	184.14	223.7
总体	231000	16	0	1246	25.46	0.00%	437.5/sec	95.58	184.14	223.7