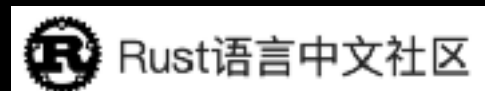


使用Redis给短链服务加速

让Rust能尽快落地到生产环境

苏林



分享内容

- 上节课开发的短链服务部署到生产，并进行压测，分析性能瓶颈的思路
- 在项目中使用 Redis 及组件介绍
- 使用 Redis 给短链 API 服务加速
- 压测带 Redis 的短链服务和原来对比
- 大型架构扩展的思路(通用思想)

期望公开课达到的目的

- 这个实战也是连续做了3次公开课了, 核心目的希望大家将Rust落地生产环境, 提升大家学习Rust的兴趣, 告别学习Rust始终在一个main文件里面写一些demo
- Rust用于web领域开发也是非常便利的
- 熟悉大型架构优化的通用手法

回顾一下上次公开课的内容

- 什么是短链(短地址)服务
- 接口定义
- 生产环境

什么是短链(短地址)服务

将长地址缩短到一个很短的地址, 用户访问这个短地址可以重定向到原本的长地址.
经常在微博、手机短信上看到比较短的URL

接口定义

接口1: `/api/create_shortlink` post请求, 将长地址转化成短地址, 并存入数据库, 接口的实现比较简单, 将长地址链接存在数据, 用数据库的自增来当成短地址

接口2: `/api/delete_shortlink` post请求, 删除短地址记录

接口3: `/api/:id` get请求, 302跳转到相应的长地址网址, 通过id查询到长地址, 并进行跳转.

接口4: `/api/not_found`, 找不到短地址对应的长地址, 跳转的链接

生产环境

配置信息

操作系统 Ubuntu Server 20.04 LTS 64位

CPU 2核

内存 4GB

公网带宽 1Mbps

<https://www.rust-lang.org/tools/install>

通过ab性能测试工具, 对线上服务进行压测

输入命令

```
ab -n 100 -c 10 http://test.com/
```

其中-n表示请求数, -c表示并发数

分析性能存在的瓶颈

Redis库介绍

<https://github.com/mitsuhiko/redis-rs>

接口代码改进

```
13 pub async fn do_redis_connect() -> Connection {
14     let client : Client = redis::Client::open( params: "redis://127.0.0.1/" ).unwrap();
15     let mut conn : RedisResult<Connection> = client.get_async_connection().await;
16     conn.unwrap()
17 }
```

```
10 pub fn app(pool: Pool<MySQL>, redis_conn: Connection) -> Router<BoxRoute> {
11     Router::new()
12         .route( path: "/", svc: get(|| async { "welcome to use axum!" }) ) : Router<Route<...>>
13         .nest( path: "/api", svc: short_links() ) : Router<Nested<...>>
14         .layer(AddExtensionLayer::new( value: pool )) : Router<Layered<AddExtension<...>>>
15         .layer(AddExtensionLayer::new( value: Arc::new( data: redis_conn )) ) : Router<Layered<AddExtension<...>>>
16         .layer(tower_http::trace::TraceLayer::new_for_http()) : Router<Layered<Trace<...>>>
17         .boxed()
18 }
```

接口代码改进

```
52 pub async fn get_shortlink(  
53     extract::Path(id): extract::Path<i32>,  
54     req: Request<Body>  
55 ) -> impl IntoResponse {  
56     let mut url : &str = "/api/not_found";  
57     let pool : &Pool<MySQL> = req.extensions().get:::<Pool<MySQL>>().unwrap();  
58     let mut conn : &Arc<Connection> = req.extensions().get:::<Arc<Connection>>().unwrap();  
59     let url: String = conn.get( key: String::from( s: "url_" ) + &*id.to_string()).unwrap();  
60     // match shortlink::get_shortlink(pool, id).await {  
61     //     Ok(record) => {  
62     //         url = Box::leak(record.url.into_boxed_str());  
63     //     }  
64     //     Err(err) => {  
65     //         println!("err = {:#?}", err);  
66     //     }  
67     // }  
68     let mut headers : HeaderMap = HeaderMap::new();  
69     headers.insert( key: LOCATION, val: url.parse().unwrap());  
70     (StatusCode::FOUND, headers, ())  
71 }
```

压测带Redis的短链服务

```
Server Software:
Server Hostname: 127.0.0.1
Server Port: 3000

Document Path: /api/25
Document Length: 0 bytes

Concurrency Level: 1000
Time taken for tests: 3.290 seconds
Complete requests: 10000
Failed requests: 0
Non-2xx responses: 10000
Total transferred: 1460000 bytes
HTML transferred: 0 bytes
Requests per second: 3039.92 [#/sec] (mean)
Time per request: 328.956 [ms] (mean)
Time per request: 0.329 [ms] (mean, across all concurrent requests)
Transfer rate: 433.43 [Kbytes/sec] received
```

Connection Times (ms)

	min	mean[+/-sd]	median	max
Connect:	0	45 203.0	0	1034
Processing:	18	255 46.6	258	394
Waiting:	1	255 46.7	258	394
Total:	37	301 209.0	260	1364

Percentage of the requests served within a certain time (ms)

50%	260
66%	283
75%	287
80%	291
90%	313
95%	344
98%	1284
99%	1311
100%	1364 (longest request)

Mysql

```
Server Software:
Server Hostname: 127.0.0.1
Server Port: 3000

Document Path: /api/10
Document Length: 0 bytes

Concurrency Level: 1000
Time taken for tests: 0.989 seconds
Complete requests: 10000
Failed requests: 0
Non-2xx responses: 10000
Total transferred: 1450000 bytes
HTML transferred: 0 bytes
Requests per second: 10113.02 [#/sec] (mean)
Time per request: 98.882 [ms] (mean)
Time per request: 0.099 [ms] (mean, across all concurrent requests)
Transfer rate: 1432.02 [Kbytes/sec] received
```

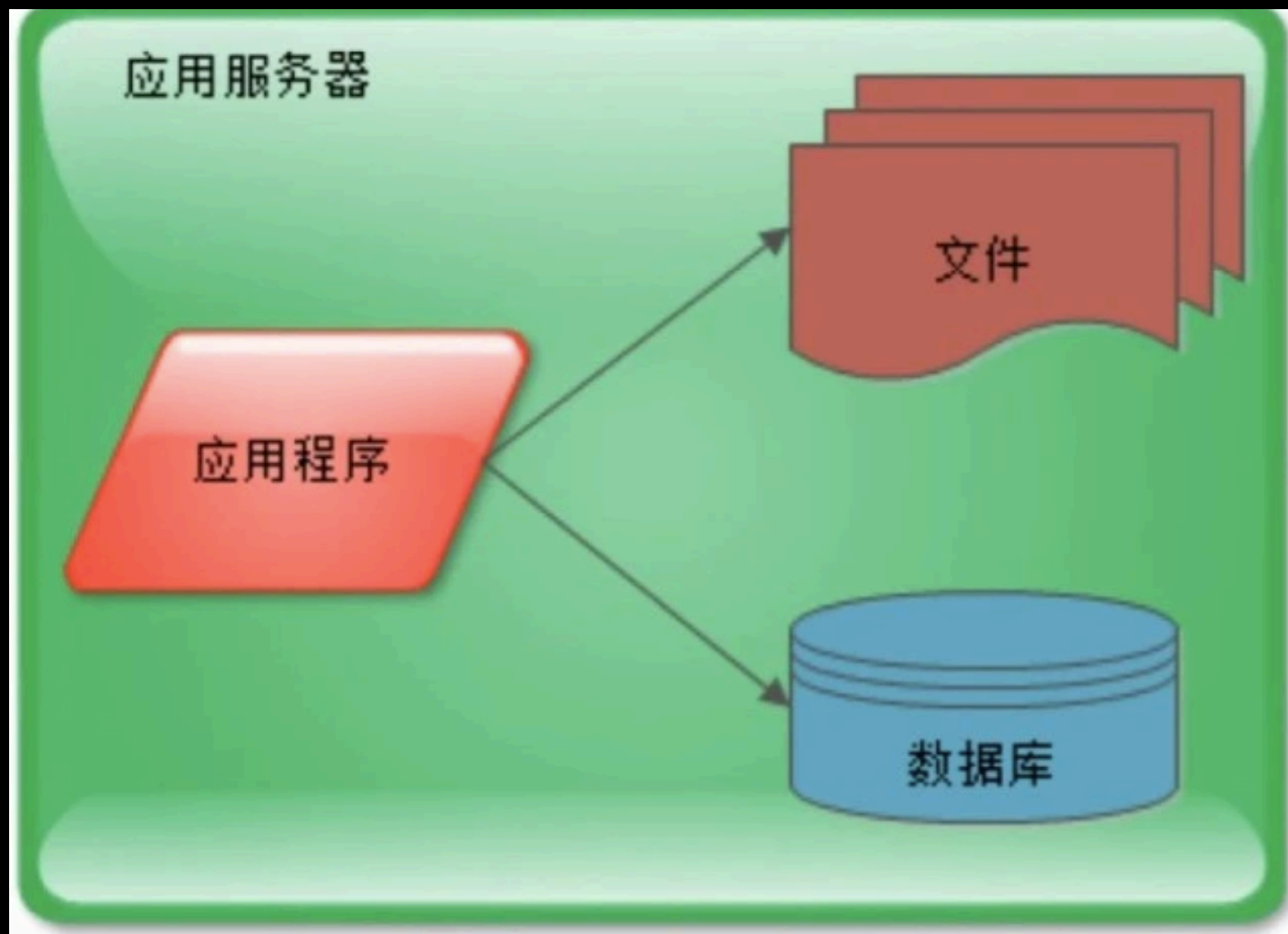
Connection Times (ms)

	min	mean[+/-sd]	median	max
Connect:	0	3 4.2	1	24
Processing:	3	25 7.3	26	58
Waiting:	0	24 7.5	26	58
Total:	7	27 6.5	28	66

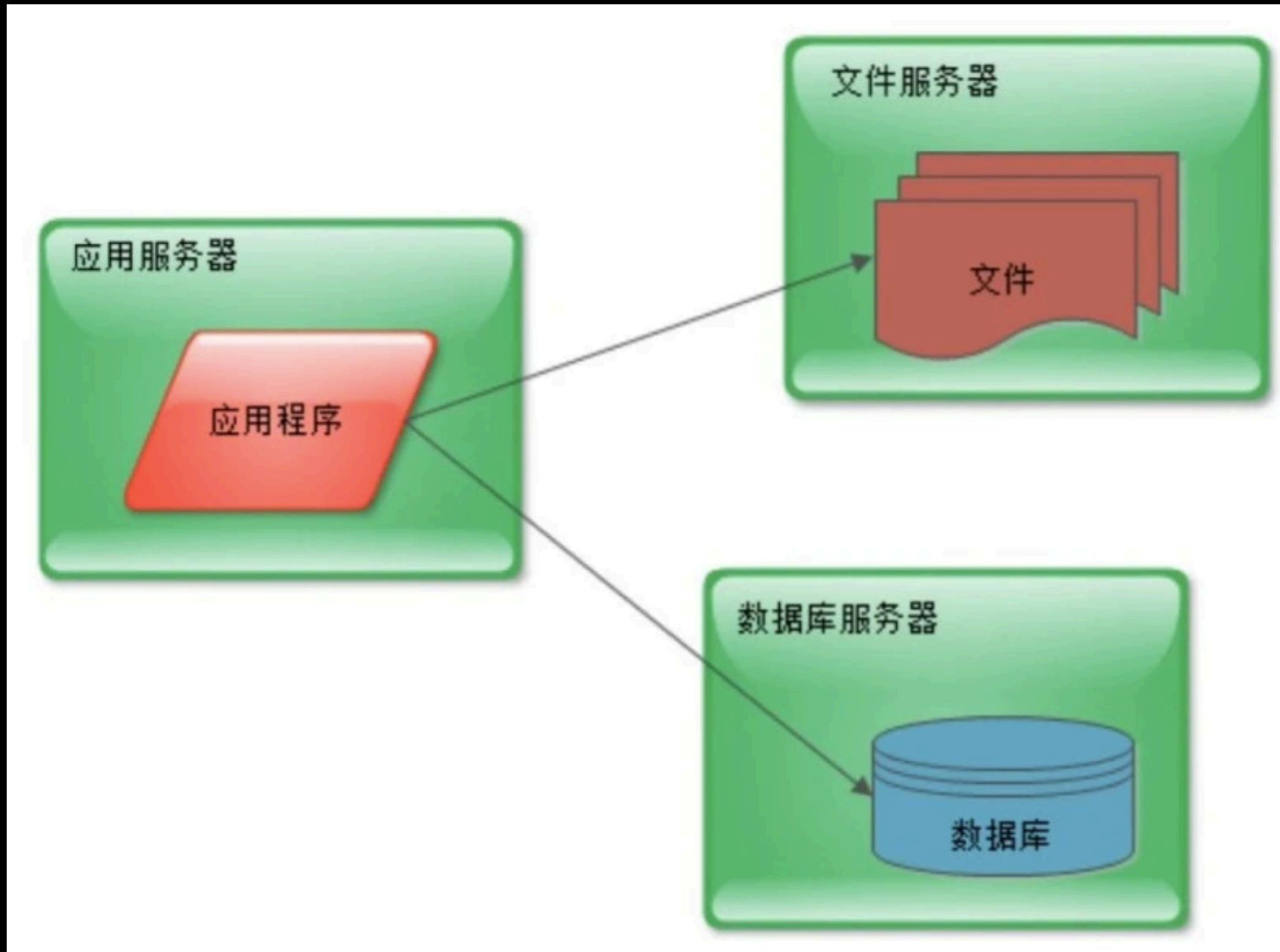
Percentage of the requests served within a certain time (ms)

50%	28
66%	29
75%	30
80%	30
90%	34
95%	39
98%	45
99%	52
100%	66 (longest request)

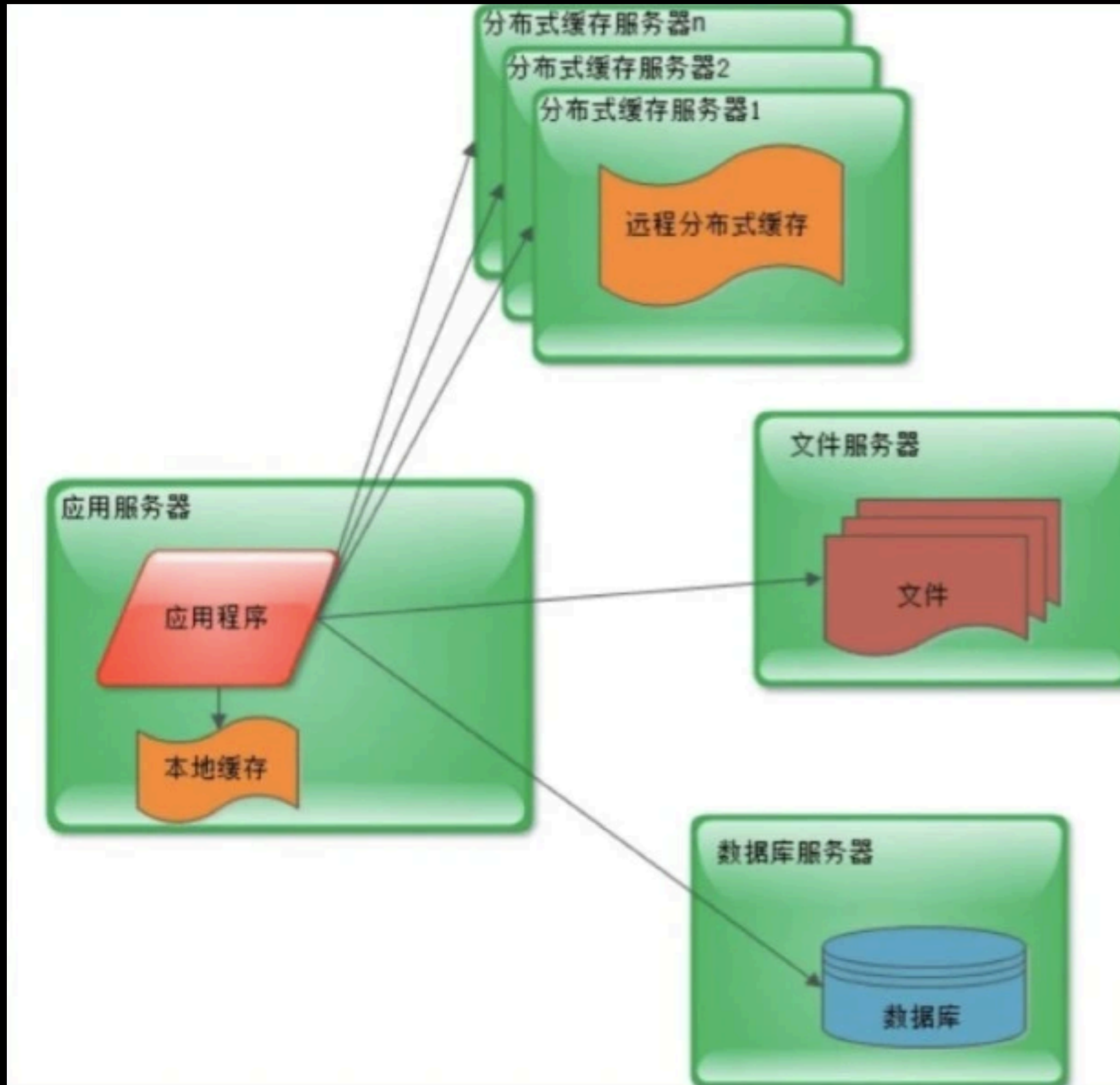
大型架构扩展的通用思路



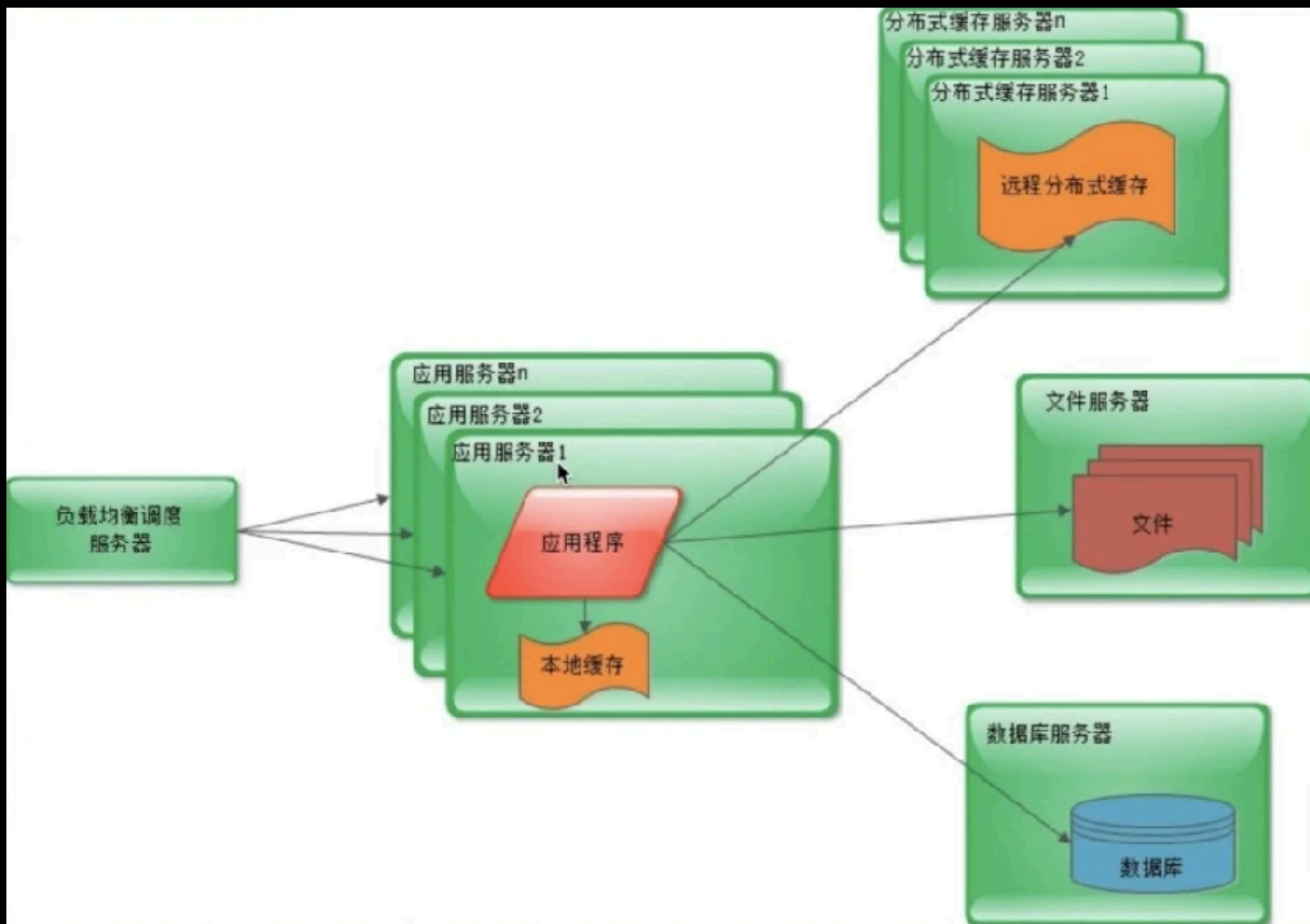
演进第一阶段: 应用、数据分离 (运维直接搞了)



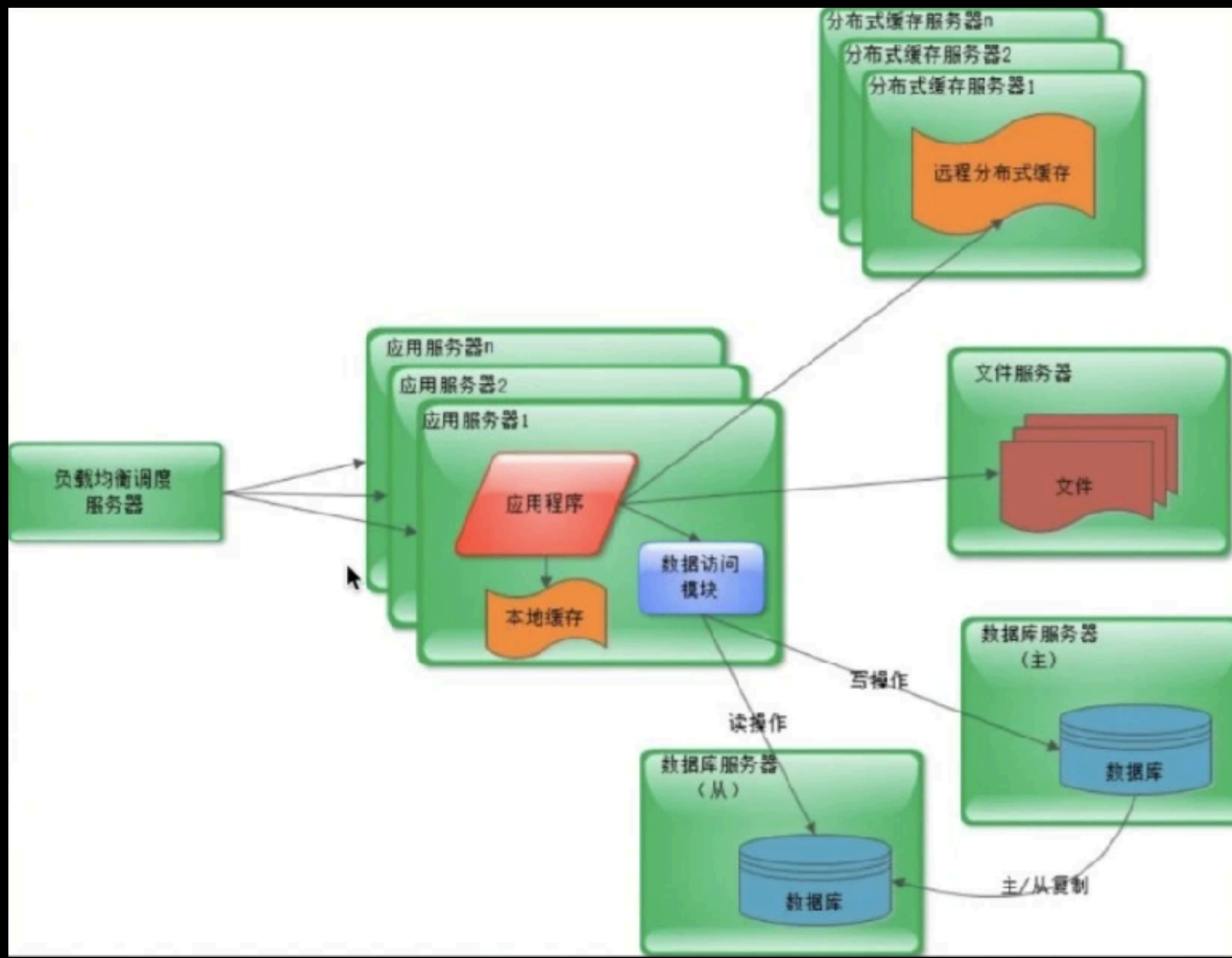
演进第二阶段: 使用缓存改善性能



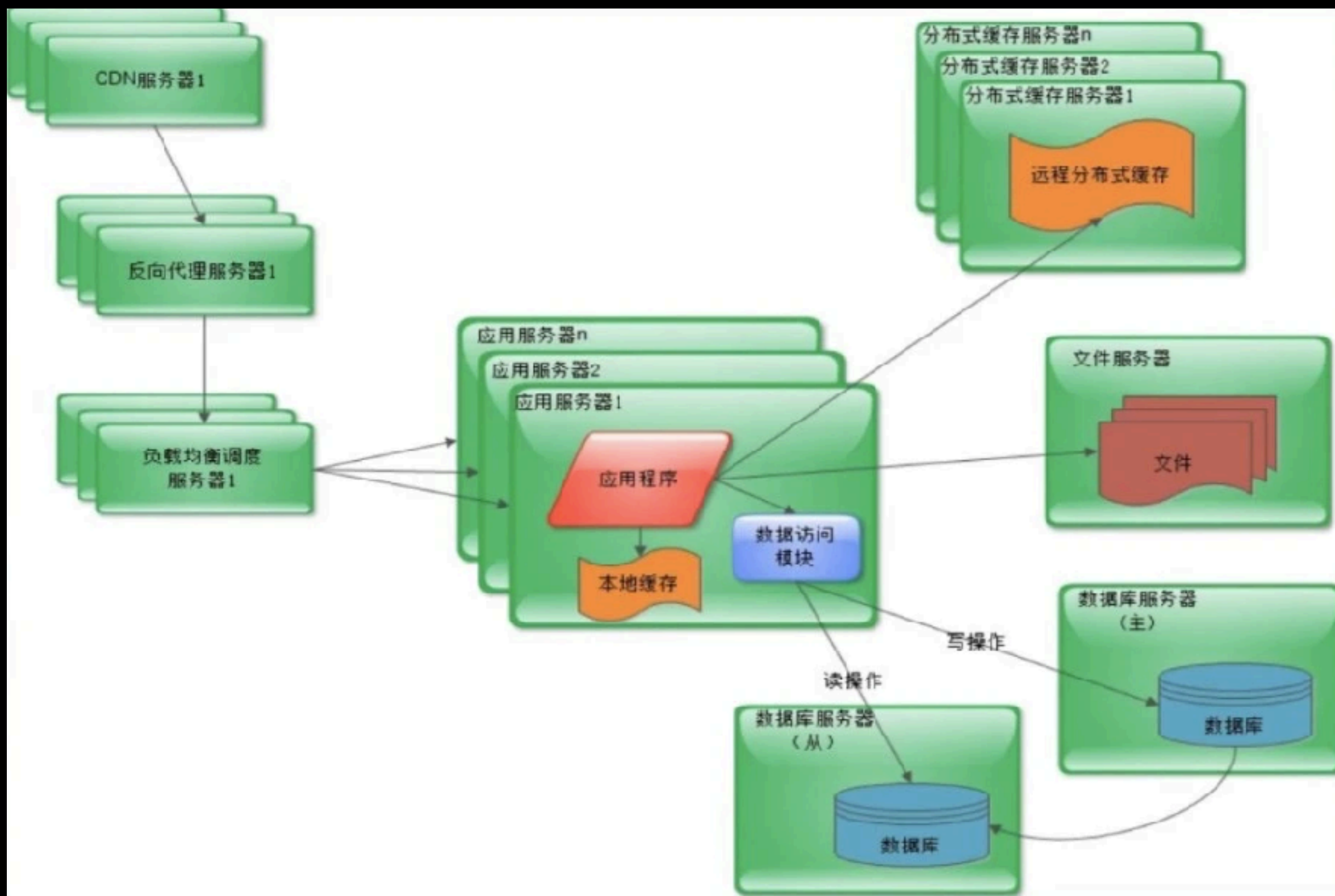
演进第三阶段: 使用应用服务器集群改善



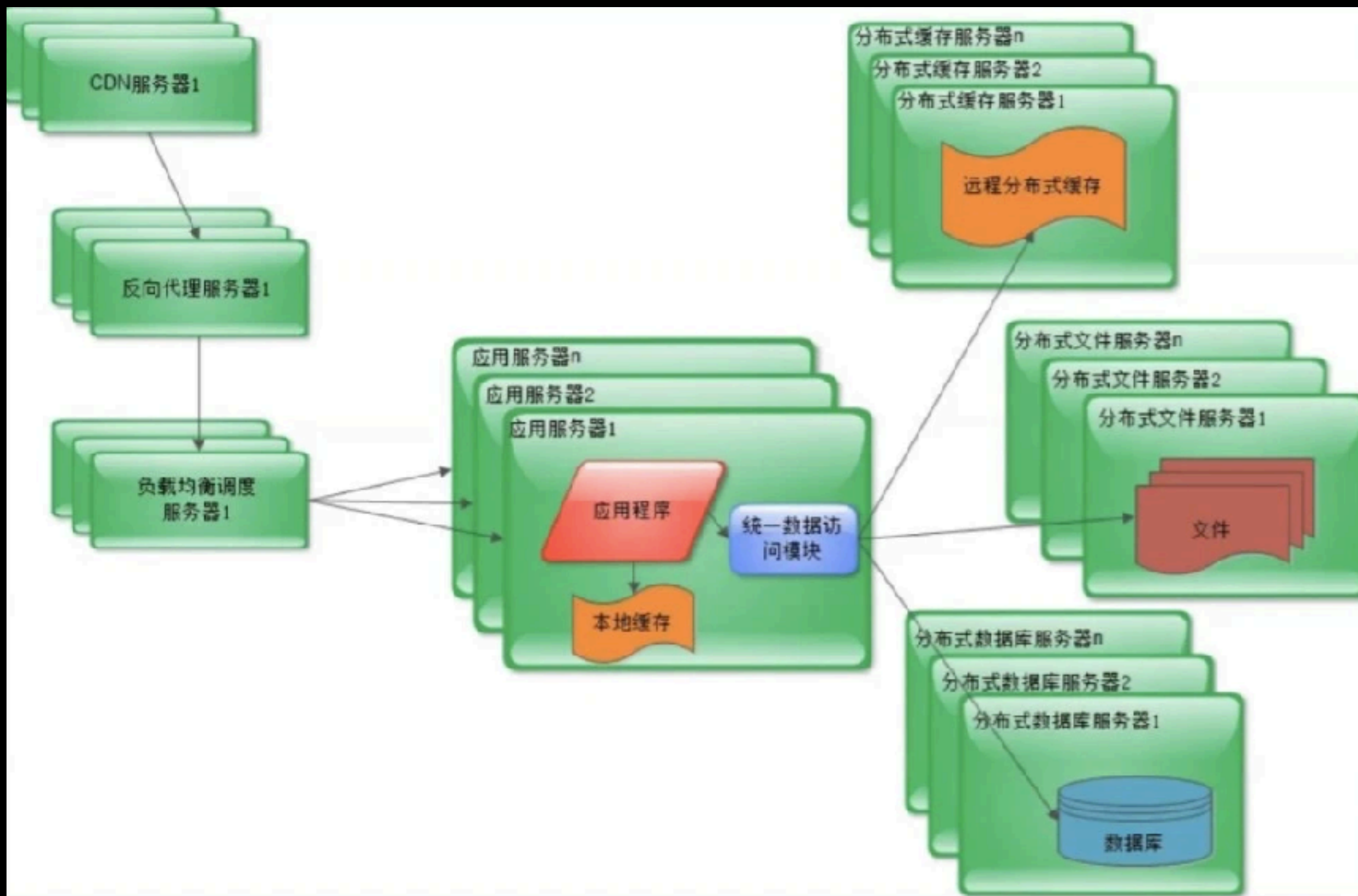
演进第四阶段: 数据库读写分离



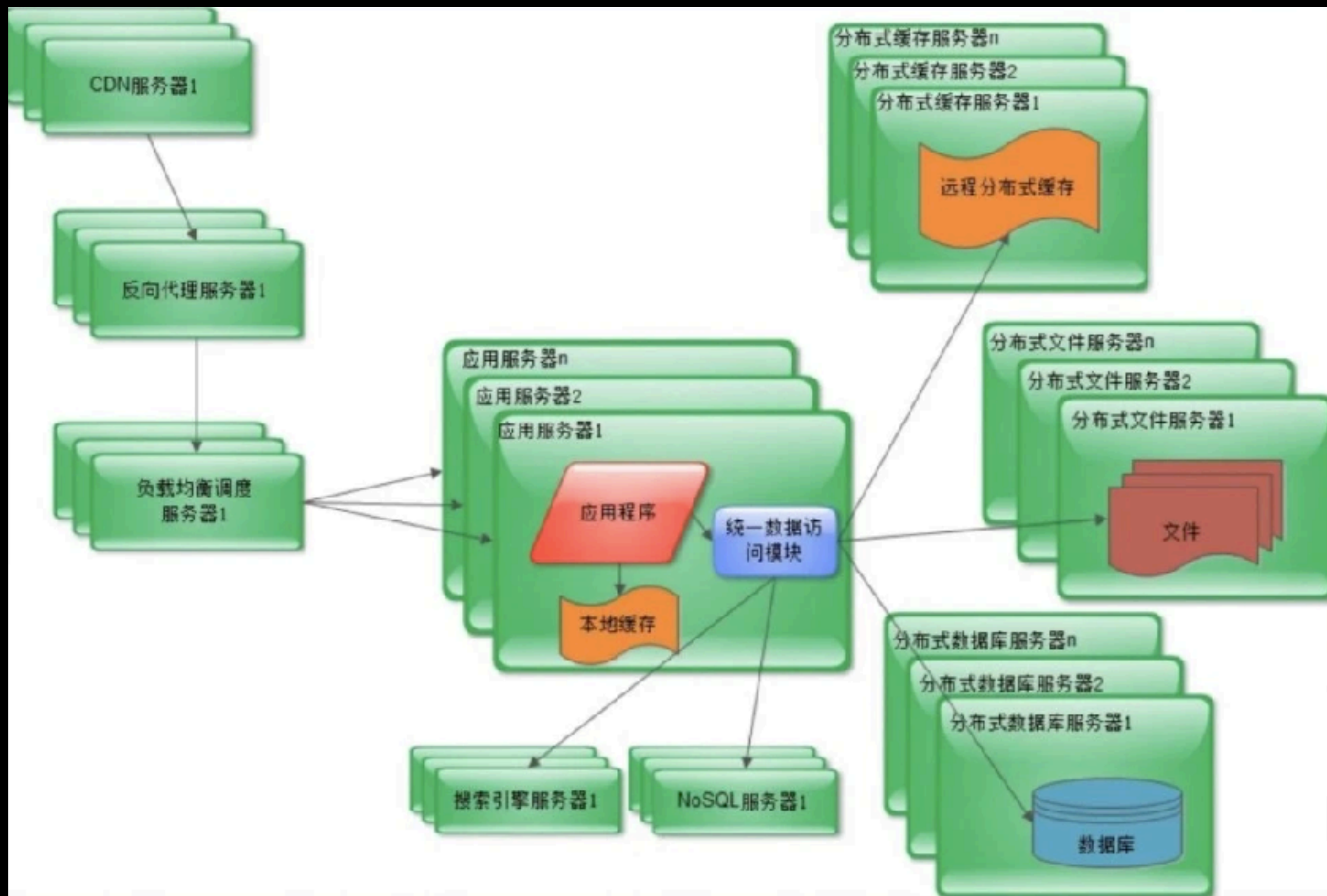
演进第五阶段: 使用反向代理和CDN加速网站



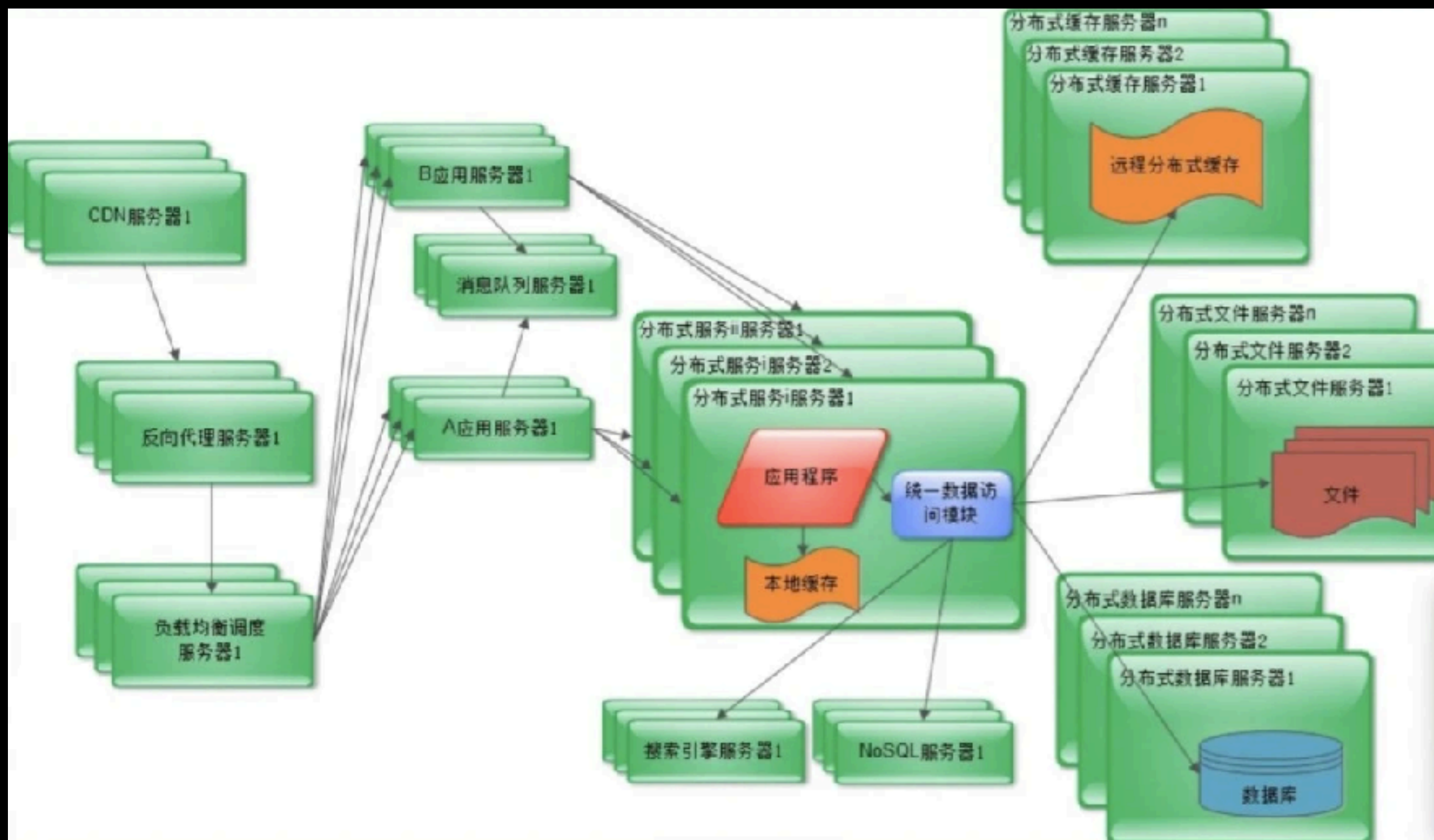
演进第六阶段: 分布式文件系统和分布式数据库



演进第七阶段: 将数据构建到搜索引擎中



演进第八阶段: 微服务及中台化



QA环节

加群一起交流Rust & Datafuse



Datafuse



Rust语言中文社区



知数堂
zhishutang.com