1. 项目概述

项目背景

随着数据量的不断增长和业务复杂度的提升，Java应用程序对大规模数据处理的需求日益迫切。传统的单机处理模式在面对海量数据时，往往会遇到内存溢出、处理速度慢等问题。因此，开发一个高效、可扩展的分布式数据处理框架具有重要的现实意义。

项目目标

- 实现一个基于Java的分布式数据处理框架
- 支持多种数据源的读取和写入
- 提供简单易用的API接口
- 具备良好的容错性和可扩展性
- 实现高效的数据处理性能

技术栈

- Java 8+
- Maven
- 网络通信：Netty
- 序列化：Protobuf
- 日志：SLF4J + Logback
- 测试：JUnit 5

2. 需求分析

功能需求

1. 数据读取
   - 支持从文件系统读取数据（本地文件、HDFS）
   - 支持从数据库读取数据（MySQL、PostgreSQL）
   - 支持从消息队列读取数据（Kafka）
2. 数据处理
   - 提供Map、Reduce、Filter等基本操作
   - 支持自定义处理函数
   - 支持数据分区和并行处理
3. 数据写入
   - 支持写入到文件系统
   - 支持写入到数据库
   - 支持写入到消息队列
4. 任务管理
   - 支持任务提交和取消
   - 支持任务状态监控
   - 支持任务失败重试

非功能需求

1. 性能
   - 单节点处理能力：至少100MB/s
   - 集群扩展性：支持线性扩展
   - 内存使用：合理控制内存占用
2. 可靠性
   - 节点故障自动恢复
   - 数据处理结果一致性保证
   - 任务执行状态持久化
3. 易用性
   - 提供简洁的API接口
   - 完善的文档和示例
   - 友好的错误提示
4. 可维护性
   - 模块化设计
   - 代码规范统一
   - 完整的单元测试

3. 系统架构

整体架构

```
+----------------+     +----------------+     +----------------+
|  Client Node   |     |  Master Node   |     |  Worker Node   |
|                |     |                |     |                |
| +------------+ |     | +------------+ |     | +------------+ |
| | Client API | |     | | Scheduler  | |     | | Executor   | |
| +------------+ |     | +------------+ |     | +------------+ |
| | Job Builder| |     | | Resource   | |     | | Task Runner| |
| +------------+ |     | | Manager    | |     | +------------+ |
|                |     | +------------+ |     | | Data Reader| |
|                |     | | Task       | |     | +------------+ |
|                |     | | Manager    | |     | | Data Writer| |
|                |     | +------------+ |     | +------------+ |
+----------------+     +----------------+     +----------------+
```

核心组件

1. Client
   - 提供用户API接口
   - 构建和提交任务
   - 监控任务执行状态
2. Master
   - 资源调度和管理
   - 任务分配和监控
   - 故障检测和恢复
3. Worker
   - 执行具体任务
   - 数据读写操作
   - 资源使用报告

通信协议

```
+--------+--------+--------+--------+
| Header | Length | Type   | Body   |
| 4 bytes| 4 bytes| 1 byte | N bytes|
+--------+--------+--------+--------+
```

- Header：魔数，用于协议识别
- Length：消息体长度
- Type：消息类型
- Body：消息内容

4. 详细设计

核心类设计

1. Job类

```java
public class Job {
    private String jobId;
    private String jobName;
    private List<Task> tasks;
    private JobConfig config;
    private JobStatus status;

    // 构建器模式
    public static class Builder {
        private Job job = new Job();

        public Builder setJobName(String name) {
            job.jobName = name;
            return this;
        }

        public Builder addTask(Task task) {
            job.tasks.add(task);
            return this;
        }

        public Job build() {
            return job;
        }
    }
}
```

2. Task类

```java
public abstract class Task {
    private String taskId;
    private TaskType type;
    private TaskStatus status;
    private int retryCount;

    public abstract void execute(TaskContext context);
}

public class MapTask extends Task {
    private MapFunction mapFunction;

    @Override
    public void execute(TaskContext context) {
        // 执行Map操作
    }
}

public class ReduceTask extends Task {
    private ReduceFunction reduceFunction;

    @Override
    public void execute(TaskContext context) {
        // 执行Reduce操作
    }
}
```

数据流设计

1. 数据分区

```
+----------+     +----------+     +----------+
| Input    |     | Input    |     | Input    |
| Split 1  |     | Split 2  |     | Split 3  |
+----------+     +----------+     +----------+
     |                |                |
     v                v                v
+----------+     +----------+     +----------+
| Map      |     | Map      |     | Map      |
| Task 1   |     | Task 2   |     | Task 3   |
+----------+     +----------+     +----------+
     |                |                |
     v                v                v
+----------+     +----------+     +----------+
| Shuffle  |     | Shuffle  |     | Shuffle  |
| Write    |     | Write    |     | Write    |
+----------+     +----------+     +----------+
     |                |                |
     +----------------+----------------+
                      |
                      v
+----------+     +----------+     +----------+
| Shuffle  |     | Shuffle  |     | Shuffle  |
| Read     |     | Read     |     | Read     |
+----------+     +----------+     +----------+
     |                |                |
     v                v                v
+----------+     +----------+     +----------+
| Reduce   |     | Reduce   |     | Reduce   |
| Task 1   |     | Task 2   |     | Task 3   |
+----------+     +----------+     +----------+
     |                |                |
     v                v                v
+----------+     +----------+     +----------+
| Output   |     | Output   |     | Output   |
| Part 1   |     | Part 2   |     | Part 3   |
+----------+     +----------+     +----------+
```

2. 数据序列化

```java
public interface Serializer {
    byte[] serialize(Object obj);
    <T> T deserialize(byte[] bytes, Class<T> clazz);
}

public class ProtobufSerializer implements Serializer {
    @Override
    public byte[] serialize(Object obj) {
        // Protobuf序列化实现
    }

    @Override
    public <T> T deserialize(byte[] bytes, Class<T> clazz) {
        // Protobuf反序列化实现
    }
}
```

容错设计

1. 任务重试

```java
public class TaskManager {
    private static final int MAX_RETRY = 3;

    public void handleTaskFailure(Task task) {
        if (task.getRetryCount() < MAX_RETRY) {
            task.incrementRetryCount();
            scheduleTask(task);
        } else {
            markTaskFailed(task);
        }
    }
}
```

2. 数据备份

```
+----------+     +----------+     +----------+
| Primary  |     | Backup   |     | Backup   |
| Data     |     | Data 1   |     | Data 2   |
+----------+     +----------+     +----------+
```

5. 数据库设计

ER图

```
+----------+       +----------+       +----------+
|  Job     |       |  Task    |       |  Worker  |
+----------+       +----------+       +----------+
| jobId    |1    n | taskId   |n    1 | workerId |
| jobName  |-------| jobId    |-------| host     |
| status   |       | type     |       | port     |
| config   |       | status   |       | status   |
| createTime|      | retryCount|      | lastHeartbeat|
+----------+       +----------+       +----------+
```

表结构

1. job表

```sql
CREATE TABLE job (
    job_id VARCHAR(64) PRIMARY KEY,
    job_name VARCHAR(128) NOT NULL,
    status VARCHAR(32) NOT NULL,
    config TEXT,
    create_time TIMESTAMP NOT NULL,
    update_time TIMESTAMP NOT NULL
);
```

2. task表

```sql
CREATE TABLE task (
    task_id VARCHAR(64) PRIMARY KEY,
    job_id VARCHAR(64) NOT NULL,
    type VARCHAR(32) NOT NULL,
    status VARCHAR(32) NOT NULL,
    retry_count INT NOT NULL DEFAULT 0,
    worker_id VARCHAR(64),
    create_time TIMESTAMP NOT NULL,
    update_time TIMESTAMP NOT NULL,
    FOREIGN KEY (job_id) REFERENCES job(job_id)
);
```

3. worker表

```sql
CREATE TABLE worker (
    worker_id VARCHAR(64) PRIMARY KEY,
    host VARCHAR(128) NOT NULL,
    port INT NOT NULL,
    status VARCHAR(32) NOT NULL,
    last_heartbeat TIMESTAMP NOT NULL
);
```

6. 接口设计

用户接口

```java
public interface JobClient {
    String submitJob(Job job);
    void cancelJob(String jobId);
    JobStatus getJobStatus(String jobId);
    List<TaskStatus> getTaskStatus(String jobId);
}
```

内部接口

```java
public interface MasterService {
    void registerWorker(WorkerInfo worker);
    void heartbeat(String workerId);
    void reportTaskStatus(String taskId, TaskStatus status);
}

public interface WorkerService {
    void executeTask(Task task);
    void cancelTask(String taskId);
    WorkerStatus getStatus();
}
```

7. 测试计划

单元测试

```java
public class JobTest {
    @Test
    public void testJobBuilder() {
        Job job = new Job.Builder()
            .setJobName("test")
            .addTask(new MapTask())
            .build();
        assertEquals("test", job.getJobName());
        assertEquals(1, job.getTasks().size());
    }
}
```

集成测试

```
+----------+     +----------+     +----------+
| Test     |     | Test     |     | Test     |
| Client   |     | Master   |     | Worker   |
+----------+     +----------+     +----------+
     |                |                |
     |  submitJob     |                |
     |--------------->|                |
     |                |  assignTask    |
     |                |--------------->|
     |                |  reportStatus  |
     |                |<---------------|
     |  getStatus     |                |
     |--------------->|                |
     |<---------------|                |
```

性能测试

1. 测试场景
   - 单节点处理能力测试
   - 集群扩展性测试
   - 内存使用测试
   - 网络带宽测试
2. 测试工具
   - JMH
   - JProfiler
   - Wireshark

8. 部署方案

环境要求

- JDK 8+
- Maven 3.6+
- MySQL 5.7+
- Kafka 2.8+

部署步骤

1. 编译打包

```
mvn clean package
```

2. 配置修改

```
# master.conf
master.host=localhost
master.port=8080
master.db.url=jdbc:mysql://localhost:3306/mydb
master.db.user=root
master.db.password=123456

# worker.conf
worker.master.host=localhost
worker.master.port=8080
worker.port=8081
```

3. 启动服务

```
# 启动Master
java -jar master.jar -c master.conf

# 启动Worker
java -jar worker.jar -c worker.conf
```

监控方案

1. 日志监控
   - 使用ELK收集日志
   - 设置关键错误告警
2. 性能监控
   - 使用Prometheus + Grafana
   - 监控CPU、内存、网络、磁盘
3. 业务监控
   - 任务执行状态
   - 数据处理量
   - 处理延迟

9. 项目计划

开发阶段

1. 第一阶段（2周）
   - 基础框架搭建
   - 核心接口设计
   - 单机版本实现
2. 第二阶段（3周）
   - 分布式功能实现
   - 容错机制实现
   - 性能优化
3. 第三阶段（2周）
   - 测试用例编写
   - 文档完善
   - 部署方案实现

里程碑

1. 2023-07-01：项目启动
2. 2023-07-15：基础框架完成
3. 2023-08-15：分布式功能完成
4. 2023-09-01：测试完成
5. 2023-09-15：项目交付

10. 风险评估

技术风险

1. 分布式一致性问题
   - 风险等级：高
   - 应对措施：采用成熟的一致性协议
2. 性能瓶颈
   - 风险等级：中
   - 应对措施：提前进行性能测试
3. 网络不稳定
   - 风险等级：中
   - 应对措施：实现重试机制

项目风险

1. 进度延期
   - 风险等级：中
   - 应对措施：合理安排开发计划
2. 资源不足
   - 风险等级：低
   - 应对措施：提前申请资源

11. 附录

参考资料

1. 《分布式系统原理与范型》
2. 《Java并发编程实战》
3. 《高性能MySQL》

术语表

术语	解释
Master	负责资源调度和任务分配的节点
Worker	执行具体任务的节点
Task	最小的执行单元
Job	由多个Task组成的完整作业
Shuffle	数据重分布过程

变更记录

版本	日期	修改人	修改内容
1.0	2023-07-01	张三	初始版本
1.1	2023-07-15	李四	添加容错设计
1.2	2023-08-01	王五	更新数据库设计

1. 项目概述

项目背景

随着互联网应用的快速发展，用户对系统响应速度的要求越来越高。缓存作为提高系统性能的重要手段，被广泛应用于各种场景。本项目旨在开发一个高性能、可扩展的缓存系统，支持多种缓存策略和分布式部署。

项目目标

- 实现一个基于内存的缓存系统
- 支持多种缓存淘汰策略（LRU、LFU、FIFO等）
- 提供简单易用的API接口
- 支持分布式部署和集群管理
- 具备良好的监控和统计功能

技术栈

- Java 11
- Maven
- 网络通信：Netty
- 序列化：Protobuf
- 日志：SLF4J + Logback
- 测试：JUnit 5 + Mockito

2. 需求分析

功能需求

1. 基本缓存操作
   - 支持get、put、delete操作
   - 支持批量操作
   - 支持过期时间设置
2. 缓存淘汰策略
   - LRU（最近最少使用）
   - LFU（最不经常使用）
   - FIFO（先进先出）
   - 支持自定义淘汰策略
3. 分布式功能
   - 支持集群部署
   - 支持数据分片
   - 支持节点故障转移
4. 监控统计
   - 缓存命中率统计
   - 内存使用监控
   - 操作耗时统计

非功能需求

1. 性能
   - 单节点QPS：至少10万
   - 平均响应时间：小于1ms
   - 内存使用：合理控制内存占用
2. 可靠性
   - 节点故障自动恢复
   - 数据一致性保证
   - 服务可用性：99.9%
3. 易用性
   - 提供简洁的API接口
   - 完善的文档和示例
   - 友好的错误提示
4. 可维护性
   - 模块化设计
   - 代码规范统一
   - 完整的单元测试

3. 系统架构

整体架构

```
+----------------+     +----------------+     +----------------+
|  Client Node   |     |  Server Node   |     |  Server Node   |
|                |     |                |     |                |
| +------------+ |     | +------------+ |     | +------------+ |
| | Client API | |     | | Cache Core | |     | | Cache Core | |
| +------------+ |     | +------------+ |     | +------------+ |
| | Connection | |     | | Network    | |     | | Network    | |
| | Pool       | |     | | Server     | |     | | Server     | |
| +------------+ |     | +------------+ |     | +------------+ |
+----------------+     +----------------+     +----------------+
```

核心组件

1. Client
   - 提供用户API接口
   - 管理连接池
   - 处理请求重试
2. Server
   - 缓存核心实现
   - 网络通信处理
   - 集群管理
3. Cache Core
   - 缓存存储
   - 淘汰策略
   - 过期处理

通信协议

```
+--------+--------+--------+--------+
| Header | Length | Type   | Body   |
| 4 bytes| 4 bytes| 1 byte | N bytes|
+--------+--------+--------+--------+
```

- Header：魔数，用于协议识别
- Length：消息体长度
- Type：消息类型
- Body：消息内容

4. 详细设计

核心类设计

1. Cache接口

```java
public interface Cache<K, V> {
    V get(K key);
    void put(K key, V value);
    void put(K key, V value, long expireTime);
    void delete(K key);
    void clear();
    int size();
    boolean contains(K key);
}
```

2. AbstractCache抽象类

```java
public abstract class AbstractCache<K, V> implements Cache<K, V> {
    protected final int maxSize;
    protected final EvictionPolicy<K, V> evictionPolicy;

    public AbstractCache(int maxSize, EvictionPolicy<K, V> evictionPolicy) {
        this.maxSize = maxSize;
        this.evictionPolicy = evictionPolicy;
    }

    protected void checkAndEvict() {
        if (size() >= maxSize) {
            evictionPolicy.evict();
        }
    }
}
```

3. LRUCache实现

```java
public class LRUCache<K, V> extends AbstractCache<K, V> {
    private final Map<K, Node<K, V>> map;
    private final Node<K, V> head;
    private final Node<K, V> tail;

    public LRUCache(int maxSize) {
        super(maxSize, new LRUEvictionPolicy<>());
        this.map = new HashMap<>();
        this.head = new Node<>();
        this.tail = new Node<>();
        head.next = tail;
        tail.prev = head;
    }

    @Override
    public V get(K key) {
        Node<K, V> node = map.get(key);
        if (node == null) {
            return null;
        }
        moveToHead(node);
        return node.value;
    }

    @Override
    public void put(K key, V value) {
        Node<K, V> node = map.get(key);
        if (node == null) {
            checkAndEvict();
            node = new Node<>(key, value);
            map.put(key, node);
            addToHead(node);
        } else {
            node.value = value;
            moveToHead(node);
        }
    }

    private void moveToHead(Node<K, V> node) {
        removeNode(node);
        addToHead(node);
    }

    private void addToHead(Node<K, V> node) {
        node.next = head.next;
        node.prev = head;
        head.next.prev = node;
        head.next = node;
    }

    private void removeNode(Node<K, V> node) {
        node.prev.next = node.next;
        node.next.prev = node.prev;
    }

    private static class Node<K, V> {
        K key;
        V value;
        Node<K, V> prev;
        Node<K, V> next;

        Node() {}

        Node(K key, V value) {
            this.key = key;
            this.value = value;
        }
    }
}
```

数据流设计

1. 写入流程

```
+----------+     +----------+     +----------+
| Client   |     | Server   |     | Cache    |
| Request  |     | Handler  |     | Core     |
+----------+     +----------+     +----------+
     |                |                |
     |  put(k,v)      |                |
     |--------------->|                |
     |                |  put(k,v)      |
     |                |--------------->|
     |                |  result        |
     |                |<---------------|
     |  response      |                |
     |<---------------|                |
```

2. 读取流程

```
+----------+     +----------+     +----------+
| Client   |     | Server   |     | Cache    |
| Request  |     | Handler  |     | Core     |
+----------+     +----------+     +----------+
     |                |                |
     |  get(k)        |                |
     |--------------->|                |
     |                |  get(k)        |
     |                |--------------->|
     |                |  value         |
     |                |<---------------|
     |  response      |                |
     |<---------------|                |
```

容错设计

1. 节点故障检测

```
+----------+     +----------+     +----------+
| Node A   |     | Node B   |     | Node C   |
| Heartbeat|     | Heartbeat|     | Heartbeat|
+----------+     +----------+     +----------+
     |                |                |
     |  ping          |                |
     |--------------->|                |
     |  pong          |                |
     |<---------------|                |
     |                |  ping          |
     |                |--------------->|
     |                |  pong          |
     |                |<---------------|
```

2. 数据备份

```
+----------+     +----------+     +----------+
| Primary  |     | Backup   |     | Backup   |
| Data     |     | Data 1   |     | Data 2   |
+----------+     +----------+     +----------+
```

5. 数据库设计

ER图

```
+----------+       +----------+       +----------+
|  Cache   |       |  Node    |       |  Config  |
+----------+       +----------+       +----------+
| key      |1    n | nodeId   |n    1 | configId |
| value    |-------| host     |-------| name     |
| expireTime|      | port     |       | value    |
| createTime|      | status   |       | updateTime|
+----------+       +----------+       +----------+
```

表结构

1. cache表

```sql
CREATE TABLE cache (
    key VARCHAR(255) PRIMARY KEY,
    value BLOB NOT NULL,
    expire_time BIGINT,
    create_time BIGINT NOT NULL
);
```

2. node表

```sql
CREATE TABLE node (
    node_id VARCHAR(64) PRIMARY KEY,
    host VARCHAR(128) NOT NULL,
    port INT NOT NULL,
    status VARCHAR(32) NOT NULL,
    last_heartbeat BIGINT NOT NULL
);
```

3. config表

```sql
CREATE TABLE config (
    config_id VARCHAR(64) PRIMARY KEY,
    name VARCHAR(128) NOT NULL,
    value TEXT NOT NULL,
    update_time BIGINT NOT NULL
);
```

6. 接口设计

用户接口

```java
public interface CacheClient {
    String get(String key);
    void put(String key, String value);
    void put(String key, String value, long expireTime);
    void delete(String key);
    void clear();
    int size();
    boolean contains(String key);
}
```

内部接口

```java
public interface ClusterManager {
    void registerNode(NodeInfo node);
    void unregisterNode(String nodeId);
    void heartbeat(String nodeId);
    List<NodeInfo> getNodes();
}

public interface DataSync {
    void syncData(String nodeId, byte[] data);
    void syncConfig(String nodeId, Config config);
}
```

7. 测试计划

单元测试

```java
public class LRUCacheTest {
    @Test
    public void testPutAndGet() {
        LRUCache<String, String> cache = new LRUCache<>(2);
        cache.put("1", "1");
        cache.put("2", "2");
        assertEquals("1", cache.get("1"));
        cache.put("3", "3");
        assertNull(cache.get("2"));
        assertEquals("3", cache.get("3"));
    }
}
```

集成测试

```
+----------+     +----------+     +----------+
| Test     |     | Test     |     | Test     |
| Client   |     | Server   |     | Cluster  |
+----------+     +----------+     +----------+
     |                |                |
     |  put(k,v)      |                |
     |--------------->|                |
     |                |  sync          |
     |                |--------------->|
     |  get(k)        |                |
     |--------------->|                |
     |<---------------|                |
```

性能测试

1. 测试场景
   - 单节点QPS测试
   - 集群扩展性测试
   - 内存使用测试
   - 网络带宽测试
2. 测试工具
   - JMH
   - JProfiler
   - Wireshark

8. 部署方案

环境要求

- JDK 11+
- Maven 3.6+
- MySQL 5.7+
- Redis 6.0+

部署步骤

1. 编译打包

```
mvn clean package
```

2. 配置修改

```
# server.conf
server.port=8080
server.host=localhost
cluster.nodes=node1:8080,node2:8080,node3:8080
```

3. 启动服务

```
java -jar cache-server.jar -c server.conf
```

监控方案

1. 日志监控
   - 使用ELK收集日志
   - 设置关键错误告警
2. 性能监控
   - 使用Prometheus + Grafana
   - 监控CPU、内存、网络、磁盘
3. 业务监控
   - 缓存命中率
   - 操作耗时
   - 错误率

9. 项目计划

开发阶段

1. 第一阶段（2周）
   - 基础框架搭建
   - 核心接口设计
   - 单机版本实现
2. 第二阶段（3周）
   - 分布式功能实现
   - 容错机制实现
   - 性能优化
3. 第三阶段（2周）
   - 测试用例编写
   - 文档完善
   - 部署方案实现

里程碑

1. 2023-07-01：项目启动
2. 2023-07-15：基础框架完成
3. 2023-08-15：分布式功能完成
4. 2023-09-01：测试完成
5. 2023-09-15：项目交付

10. 风险评估

技术风险

1. 分布式一致性问题
   - 风险等级：高
   - 应对措施：采用成熟的一致性协议
2. 性能瓶颈
   - 风险等级：中
   - 应对措施：提前进行性能测试
3. 网络不稳定
   - 风险等级：中
   - 应对措施：实现重试机制

项目风险

1. 进度延期
   - 风险等级：中
   - 应对措施：合理安排开发计划
2. 资源不足
   - 风险等级：低
   - 应对措施：提前申请资源

11. 附录

参考资料

1. 《分布式系统原理与范型》
2. 《Java并发编程实战》
3. 《高性能MySQL》

术语表

术语	解释
LRU	最近最少使用
LFU	最不经常使用
FIFO	先进先出
QPS	每秒查询数
TTL	生存时间

变更记录

版本	日期	修改人	修改内容
1.0	2023-07-01	张三	初始版本
1.1	2023-07-15	李四	添加容错设计
1.2	2023-08-01	王五	更新数据库设计

1. 项目概述

项目背景

随着互联网应用的快速发展，用户对系统响应速度的要求越来越高。缓存作为提高系统性能的重要手段，被广泛应用于各种场景。本项目旨在开发一个高性能、可扩展的缓存系统，支持多种缓存策略和分布式部署。

项目目标

- 实现一个基于内存的缓存系统
- 支持多种缓存淘汰策略（LRU、LFU、FIFO等）
- 提供简单易用的API接口
- 支持分布式部署和集群管理
- 具备良好的监控和统计功能

技术栈

- Java 11
- Maven
- 网络通信：Netty
- 序列化：Protobuf
- 日志：SLF4J + Logback
- 测试：JUnit 5 + Mockito

2. 需求分析

功能需求

1. 基本缓存操作
   - 支持get、put、delete操作
   - 支持批量操作
   - 支持过期时间设置
2. 缓存淘汰策略
   - LRU（最近最少使用）
   - LFU（最不经常使用）
   - FIFO（先进先出）
   - 支持自定义淘汰策略
3. 分布式功能
   - 支持集群部署
   - 支持数据分片
   - 支持节点故障转移
4. 监控统计
   - 缓存命中率统计
   - 内存使用监控
   - 操作耗时统计

非功能需求

1. 性能
   - 单节点QPS：至少10万
   - 平均响应时间：小于1ms
   - 内存使用：合理控制内存占用
2. 可靠性
   - 节点故障自动恢复
   - 数据一致性保证
   - 服务可用性：99.9%
3. 易用性
   - 提供简洁的API接口
   - 完善的文档和示例
   - 友好的错误提示
4. 可维护性
   - 模块化设计
   - 代码规范统一
   - 完整的单元测试

3. 系统架构

整体架构

```
+----------------+     +----------------+     +----------------+
|  Client Node   |     |  Server Node   |     |  Server Node   |
|                |     |                |     |                |
| +------------+ |     | +------------+ |     | +------------+ |
| | Client API | |     | | Cache Core | |     | | Cache Core | |
| +------------+ |     | +------------+ |     | +------------+ |
| | Connection | |     | | Network    | |     | | Network    | |
| | Pool       | |     | | Server     | |     | | Server     | |
| +------------+ |     | +------------+ |     | +------------+ |
+----------------+     +----------------+     +----------------+
```

核心组件

1. Client
   - 提供用户API接口
   - 管理连接池
   - 处理请求重试
2. Server
   - 缓存核心实现
   - 网络通信处理
   - 集群管理
3. Cache Core
   - 缓存存储
   - 淘汰策略
   - 过期处理

通信协议

```
+--------+--------+--------+--------+
| Header | Length | Type   | Body   |
| 4 bytes| 4 bytes| 1 byte | N bytes|
+--------+--------+--------+--------+
```

- Header：魔数，用于协议识别
- Length：消息体长度
- Type：消息类型
- Body：消息内容

4. 详细设计

核心类设计

1. Cache接口

```java
public interface Cache<K, V> {
    V get(K key);
    void put(K key, V value);
    void put(K key, V value, long expireTime);
    void delete(K key);
    void clear();
    int size();
    boolean contains(K key);
}
```

2. AbstractCache抽象类

```java
public abstract class AbstractCache<K, V> implements Cache<K, V> {
    protected final int maxSize;
    protected final EvictionPolicy<K, V> evictionPolicy;

    public AbstractCache(int maxSize, EvictionPolicy<K, V> evictionPolicy) {
        this.maxSize = maxSize;
        this.evictionPolicy = evictionPolicy;
    }

    protected void checkAndEvict() {
        if (size() >= maxSize) {
            evictionPolicy.evict();
        }
    }
}
```

3. LRUCache实现

```java
public class LRUCache<K, V> extends AbstractCache<K, V> {
    private final Map<K, Node<K, V>> map;
    private final Node<K, V> head;
    private final Node<K, V> tail;

    public LRUCache(int maxSize) {
        super(maxSize, new LRUEvictionPolicy<>());
        this.map = new HashMap<>();
        this.head = new Node<>();
        this.tail = new Node<>();
        head.next = tail;
        tail.prev = head;
    }

    @Override
    public V get(K key) {
        Node<K, V> node = map.get(key);
        if (node == null) {
            return null;
        }
        moveToHead(node);
        return node.value;
    }

    @Override
    public void put(K key, V value) {
        Node<K, V> node = map.get(key);
        if (node == null) {
            checkAndEvict();
            node = new Node<>(key, value);
            map.put(key, node);
            addToHead(node);
        } else {
            node.value = value;
            moveToHead(node);
        }
    }

    private void moveToHead(Node<K, V> node) {
        removeNode(node);
        addToHead(node);
    }

    private void addToHead(Node<K, V> node) {
        node.next = head.next;
        node.prev = head;
        head.next.prev = node;
        head.next = node;
    }

    private void removeNode(Node<K, V> node) {
        node.prev.next = node.next;
        node.next.prev = node.prev;
    }

    private static class Node<K, V> {
        K key;
        V value;
        Node<K, V> prev;
        Node<K, V> next;

        Node() {}

        Node(K key, V value) {
            this.key = key;
            this.value = value;
        }
    }
}
```

数据流设计

1. 写入流程

```
+----------+     +----------+     +----------+
| Client   |     | Server   |     | Cache    |
| Request  |     | Handler  |     | Core     |
+----------+     +----------+     +----------+
     |                |                |
     |  put(k,v)      |                |
     |--------------->|                |
     |                |  put(k,v)      |
     |                |--------------->|
     |                |  result        |
     |                |<---------------|
     |  response      |                |
     |<---------------|                |
```

2. 读取流程

```
+----------+     +----------+     +----------+
| Client   |     | Server   |     | Cache    |
| Request  |     | Handler  |     | Core     |
+----------+     +----------+     +----------+
     |                |                |
     |  get(k)        |                |
     |--------------->|                |
     |                |  get(k)        |
     |                |--------------->|
     |                |  value         |
     |                |<---------------|
     |  response      |                |
     |<---------------|                |
```

容错设计

1. 节点故障检测

```
+----------+     +----------+     +----------+
| Node A   |     | Node B   |     | Node C   |
| Heartbeat|     | Heartbeat|     | Heartbeat|
+----------+     +----------+     +----------+
     |                |                |
     |  ping          |                |
     |--------------->|                |
     |  pong          |                |
     |<---------------|                |
     |                |  ping          |
     |                |--------------->|
     |                |  pong          |
     |                |<---------------|
```

2. 数据备份

```
+----------+     +----------+     +----------+
| Primary  |     | Backup   |     | Backup   |
| Data     |     | Data 1   |     | Data 2   |
+----------+     +----------+     +----------+
```

5. 数据库设计

ER图

```
+----------+       +----------+       +----------+
|  Cache   |       |  Node    |       |  Config  |
+----------+       +----------+       +----------+
| key      |1    n | nodeId   |n    1 | configId |
| value    |-------| host     |-------| name     |
| expireTime|      | port     |       | value    |
| createTime|      | status   |       | updateTime|
+----------+       +----------+       +----------+
```

表结构

1. cache表

```sql
CREATE TABLE cache (
    key VARCHAR(255) PRIMARY KEY,
    value BLOB NOT NULL,
    expire_time BIGINT,
    create_time BIGINT NOT NULL
);
```

2. node表

```sql
CREATE TABLE node (
    node_id VARCHAR(64) PRIMARY KEY,
    host VARCHAR(128) NOT NULL,
    port INT NOT NULL,
    status VARCHAR(32) NOT NULL,
    last_heartbeat BIGINT NOT NULL
);
```

3. config表

```sql
CREATE TABLE config (
    config_id VARCHAR(64) PRIMARY KEY,
    name VARCHAR(128) NOT NULL,
    value TEXT NOT NULL,
    update_time BIGINT NOT NULL
);
```

6. 接口设计

用户接口

```java
public interface CacheClient {
    String get(String key);
    void put(String key, String value);
    void put(String key, String value, long expireTime);
    void delete(String key);
    void clear();
    int size();
    boolean contains(String key);
}
```

内部接口

```java
public interface ClusterManager {
    void registerNode(NodeInfo node);
    void unregisterNode(String nodeId);
    void heartbeat(String nodeId);
    List<NodeInfo> getNodes();
}

public interface DataSync {
    void syncData(String nodeId, byte[] data);
    void syncConfig(String nodeId, Config config);
}
```

7. 测试计划

单元测试

```java
public class LRUCacheTest {
    @Test
    public void testPutAndGet() {
        LRUCache<String, String> cache = new LRUCache<>(2);
        cache.put("1", "1");
        cache.put("2", "2");
        assertEquals("1", cache.get("1"));
        cache.put("3", "3");
        assertNull(cache.get("2"));
        assertEquals("3", cache.get("3"));
    }
}
```

集成测试

```
+----------+     +----------+     +----------+
| Test     |     | Test     |     | Test     |
| Client   |     | Server   |     | Cluster  |
+----------+     +----------+     +----------+
     |                |                |
     |  put(k,v)      |                |
     |--------------->|                |
     |                |  sync          |
     |                |--------------->|
     |  get(k)        |                |
     |--------------->|                |
     |<---------------|                |
```

性能测试

1. 测试场景
   - 单节点QPS测试
   - 集群扩展性测试
   - 内存使用测试
   - 网络带宽测试
2. 测试工具
   - JMH
   - JProfiler
   - Wireshark

8. 部署方案

环境要求

- JDK 11+
- Maven 3.6+
- MySQL 5.7+
- Redis 6.0+

部署步骤

1. 编译打包

```
mvn clean package
```

2. 配置修改

```
# server.conf
server.port=8080
server.host=localhost
cluster.nodes=node1:8080,node2:8080,node3:8080
```

3. 启动服务

```
java -jar cache-server.jar -c server.conf
```

监控方案

1. 日志监控
   - 使用ELK收集日志
   - 设置关键错误告警
2. 性能监控
   - 使用Prometheus + Grafana
   - 监控CPU、内存、网络、磁盘
3. 业务监控
   - 缓存命中率
   - 操作耗时
   - 错误率

9. 项目计划

开发阶段

1. 第一阶段（2周）
   - 基础框架搭建
   - 核心接口设计
   - 单机版本实现
2. 第二阶段（3周）
   - 分布式功能实现
   - 容错机制实现
   - 性能优化
3. 第三阶段（2周）
   - 测试用例编写
   - 文档完善
   - 部署方案实现

里程碑

1. 2023-07-01：项目启动
2. 2023-07-15：基础框架完成
3. 2023-08-15：分布式功能完成
4. 2023-09-01：测试完成
5. 2023-09-15：项目交付

10. 风险评估

技术风险

1. 分布式一致性问题
   - 风险等级：高
   - 应对措施：采用成熟的一致性协议
2. 性能瓶颈
   - 风险等级：中
   - 应对措施：提前进行性能测试
3. 网络不稳定
   - 风险等级：中
   - 应对措施：实现重试机制

项目风险

1. 进度延期
   - 风险等级：中
   - 应对措施：合理安排开发计划
2. 资源不足
   - 风险等级：低
   - 应对措施：提前申请资源

11. 附录

参考资料

1. 《分布式系统原理与范型》
2. 《Java并发编程实战》
3. 《高性能MySQL》

术语表

术语	解释
LRU	最近最少使用
LFU	最不经常使用
FIFO	先进先出
QPS	每秒查询数
TTL	生存时间

变更记录

版本	日期	修改人	修改内容
1.0	2023-07-01	张三	初始版本
1.1	2023-07-15	李四	添加容错设计
1.2	2023-08-01	王五	更新数据库设计

1. 项目概述

项目背景

随着互联网应用的快速发展，用户对系统响应速度的要求越来越高。缓存作为提高系统性能的重要手段，被广泛应用于各种场景。本项目旨在开发一个高性能、可扩展的缓存系统，支持多种缓存策略和分布式部署。

项目目标

- 实现一个基于内存的缓存系统
- 支持多种缓存淘汰策略（LRU、LFU、FIFO等）
- 提供简单易用的API接口
- 支持分布式部署和集群管理
- 具备良好的监控和统计功能

技术栈

- Java 11
- Maven
- 网络通信：Netty
- 序列化：Protobuf
- 日志：SLF4J + Logback
- 测试：JUnit 5 + Mockito

2. 需求分析

功能需求

1. 基本缓存操作
   - 支持get、put、delete操作
   - 支持批量操作
   - 支持过期时间设置
2. 缓存淘汰策略
   - LRU（最近最少使用）
   - LFU（最不经常使用）
   - FIFO（先进先出）
   - 支持自定义淘汰策略
3. 分布式功能
   - 支持集群部署
   - 支持数据分片
   - 支持节点故障转移
4. 监控统计
   - 缓存命中率统计
   - 内存使用监控
   - 操作耗时统计

非功能需求

1. 性能
   - 单节点QPS：至少10万
   - 平均响应时间：小于1ms
   - 内存使用：合理控制内存占用
2. 可靠性
   - 节点故障自动恢复
   - 数据一致性保证
   - 服务可用性：99.9%
3. 易用性
   - 提供简洁的API接口
   - 完善的文档和示例
   - 友好的错误提示
4. 可维护性
   - 模块化设计
   - 代码规范统一
   - 完整的单元测试

3. 系统架构

整体架构

```
+----------------+     +----------------+     +----------------+
|  Client Node   |     |  Server Node   |     |  Server Node   |
|                |     |                |     |                |
| +------------+ |     | +------------+ |     | +------------+ |
| | Client API | |     | | Cache Core | |     | | Cache Core | |
| +------------+ |     | +------------+ |     | +------------+ |
| | Connection | |     | | Network    | |     | | Network    | |
| | Pool       | |     | | Server     | |     | | Server     | |
| +------------+ |     | +------------+ |     | +------------+ |
+----------------+     +----------------+     +----------------+
```

核心组件

1. Client
   - 提供用户API接口
   - 管理连接池
   - 处理请求重试
2. Server
   - 缓存核心实现
   - 网络通信处理
   - 集群管理
3. Cache Core
   - 缓存存储
   - 淘汰策略
   - 过期处理

通信协议

```
+--------+--------+--------+--------+
| Header | Length | Type   | Body   |
| 4 bytes| 4 bytes| 1 byte | N bytes|
+--------+--------+--------+--------+
```

- Header：魔数，用于协议识别
-