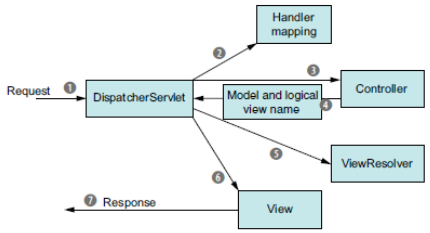
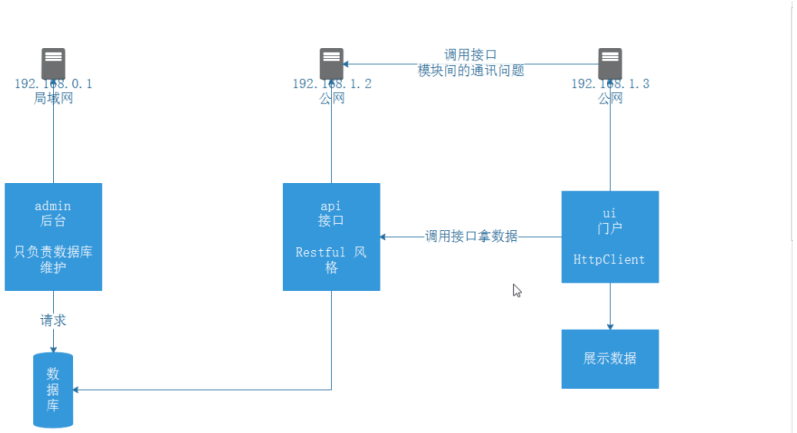
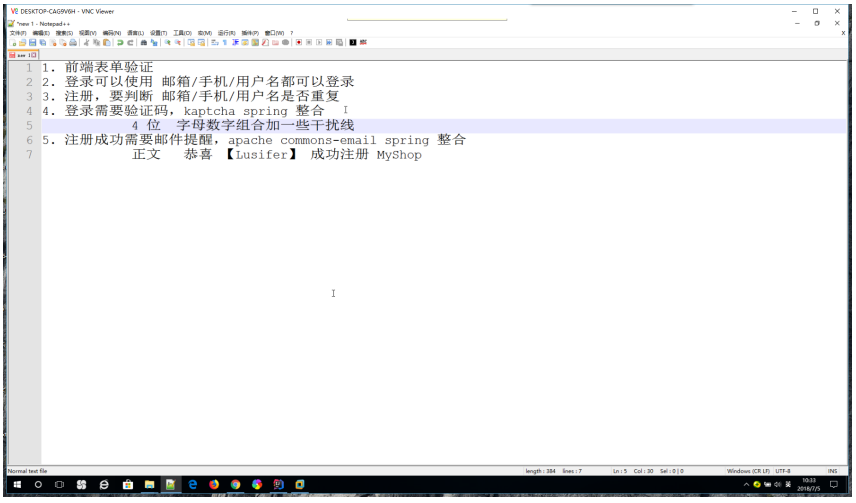
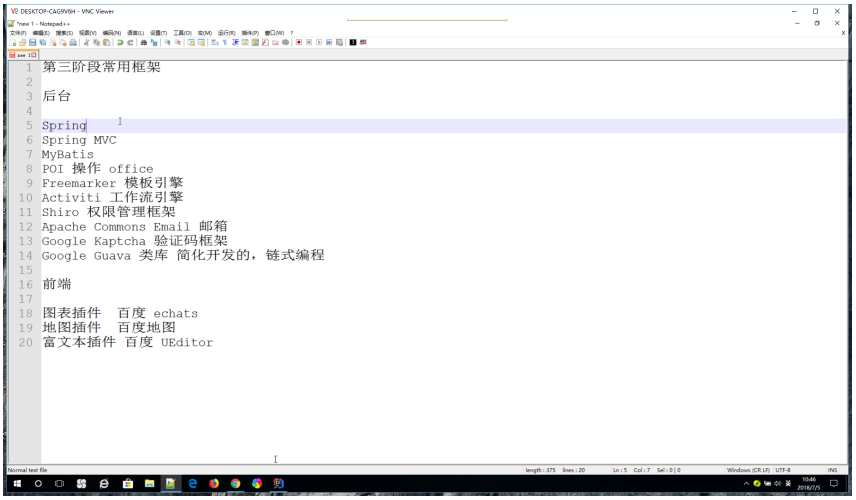


Figure 5.1 A request couriers information to several stops on its way to producing the desired results.



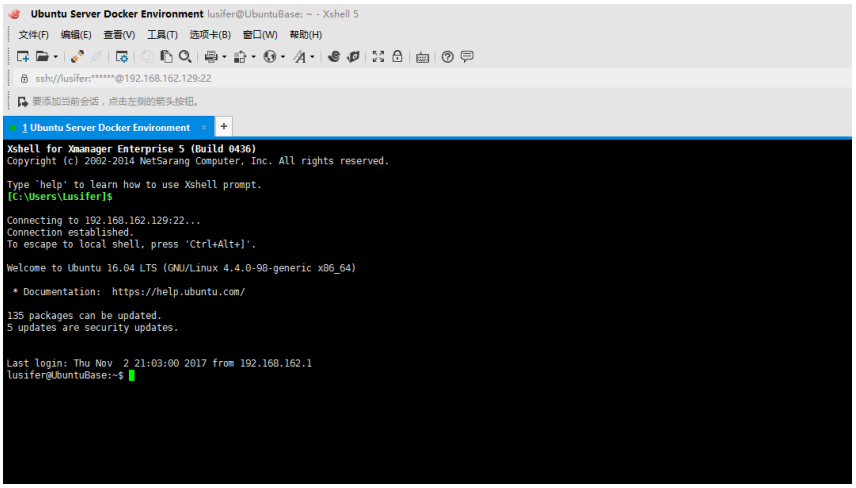

```
API 文档

请求地址：http://localhost:8081/content/findContentByCategoryId
请求方式：POST
请求参数：category_id 类目 ID  必须  数字
响应结果：

[
    {
        "id": 33,
        "title": "ad1",
        "subTitle": "ad1",
        "titleDesc": "ad1",
        "url": "https://sale.jd.com/act/XkCzhoisOMSW.html",
        "pic": "https://m.360buyimg.com/babel/jfs/t20164/187/1771326168/92964/b42fade7/5b359ab2N93be3a65.jp
        "pic2": ""
    }
]

category_id 类目 ID  必须  数字

[
    {
        "id": 33,
        "title": "ad1",
        "subTitle": "ad1",
```

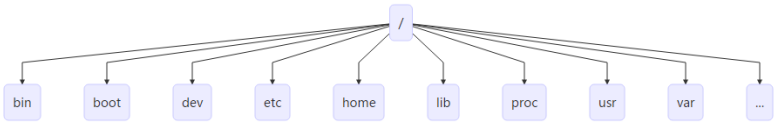




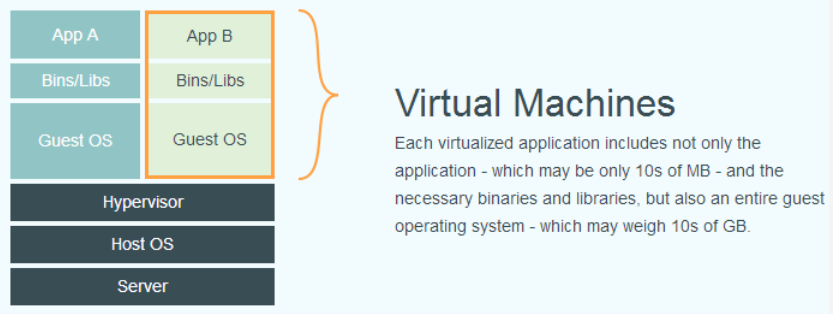
目前国内 Linux 更多的是应用于服务器上，而桌面操作系统更多使用的是 Windows。主要区别如下

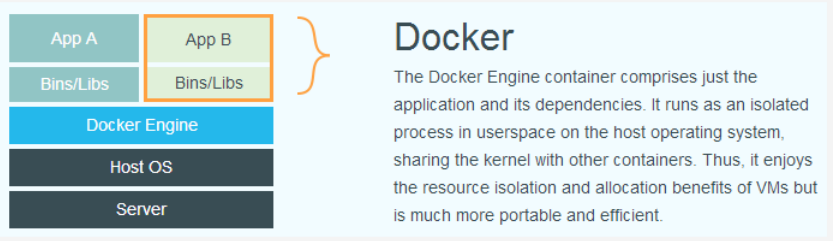
比较	Windows	Linux
界面	界面统一，外壳程序固定所有 Windows 程序菜单几乎一致，快捷键也几乎相同	图形界面风格依发布版不同而不同，可能互不兼容。GNU/Linux 的终端机是从 UNIX 传承下来，基本命令和操作方法也几乎一致。
驱动程序	驱动程序丰富，版本更新频繁。默认安装程序里面一般包含有该版本发布时流行的硬件驱动程序，之后所出的新硬件驱动依赖于硬件厂商提供。对于一些老硬件，如果没有了原配的驱动有时很难支持。另外，有时硬件厂商未提供所需版本的 Windows 下的驱动，也会比较头痛。	由志愿者开发，由Linux核心开发小组发布，很多硬件厂商基于版权考虑并未提供驱动程序，尽管多数无需手动安装，但是涉及安装则相对复杂，使得新用户面对驱动程序问题（是否存在和安装方法）会一筹莫展。但是在开源开发模式下，许多老硬件尽管在 Windows 下很难支持的也容易找到驱动。HP、Intel、AMD 等硬件厂商逐步不同程度支持开源驱动，问题正在得到缓解。
使用	使用比较简单，容易入门。图形化界面对没有计算机背景知识的用户使用十分有利。	图形界面使用简单，容易入门。文字界面，需要学习才能掌握。
学习	系统构造复杂、变化频繁，且知识、技能淘汰快，深入学习困难。	系统构造简单、稳定，且知识、技能传承性好，深入学习相对容易。
软件	每一种特定功能可能都需要商业软件的支持，需要购买相应的授权。	大部分软件都可以自由获取，同样功能的软件选择较少。



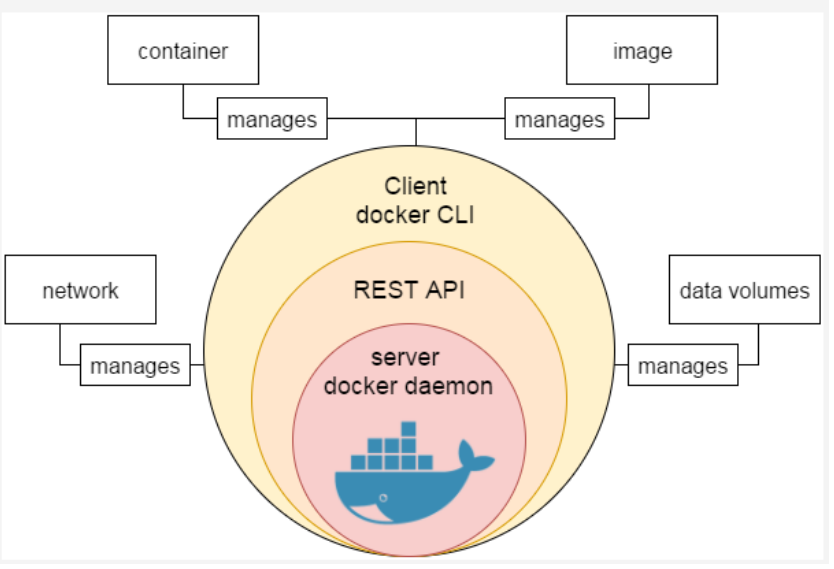


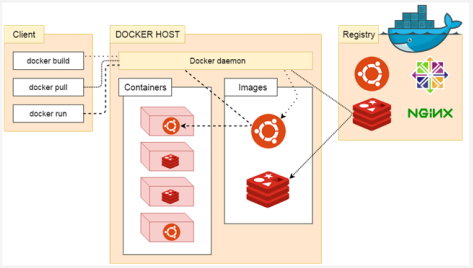
目录	说明
bin	存放二进制可执行文件(ls,cat,mkdir等)
boot	存放用于系统引导时使用的各种文件
dev	用于存放设备文件
etc	存放系统配置文件
home	存放所有用户文件的根目录
lib	存放跟文件系统中的程序运行所需要的共享库及内核模块
mnt	系统管理员安装临时文件系统的安装点
opt	额外安装的可选应用程序包所放置的位置
proc	虚拟文件系统，存放当前内存的映射
root	超级用户目录
sbin	存放二进制可执行文件，只有root才能访问
tmp	用于存放各种临时文件
usr	用于存放系统应用程序，比较重要的目录/usr/local 本地管理员软件安装目录
var	用于存放运行时需要改变数据的文件



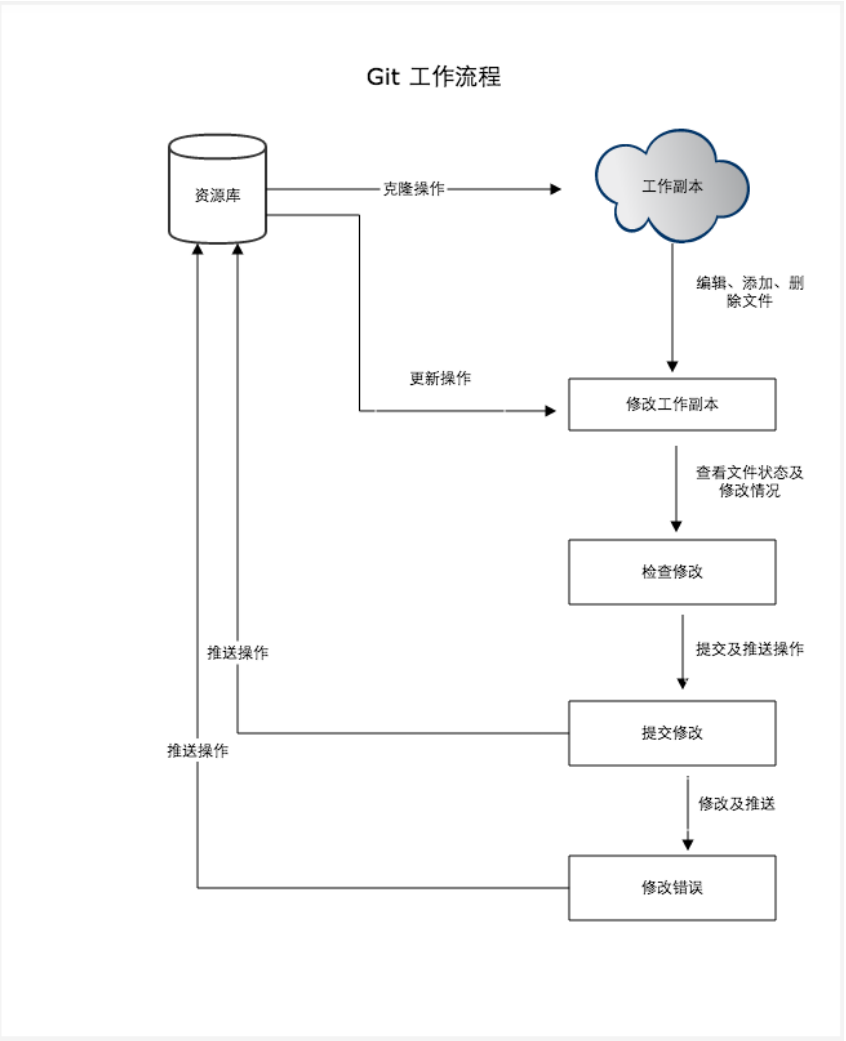


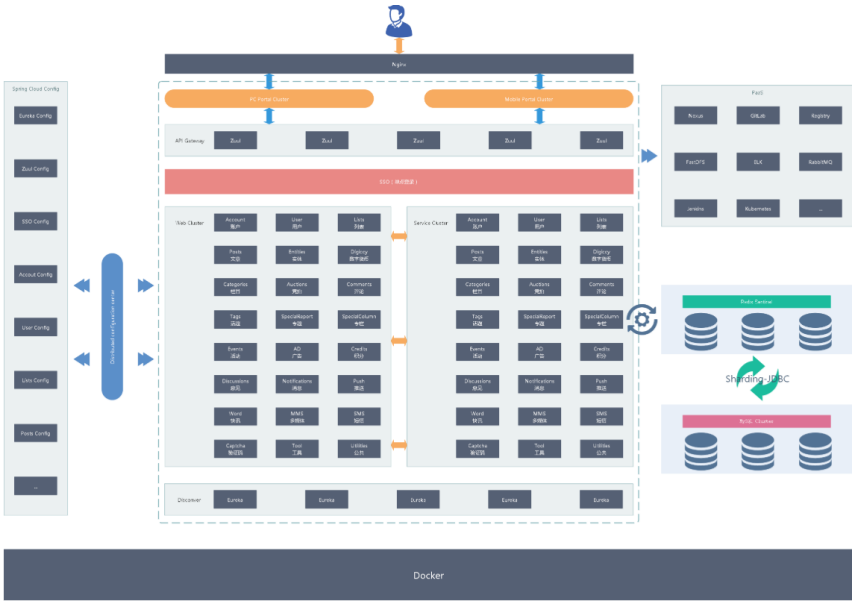
特性	容器	虚拟机
启动	秒级	分钟级
硬盘使用	一般为 MB	一般为 GB
性能	接近原生	弱于
系统支持量	单机支持上千个容器	一般几十个

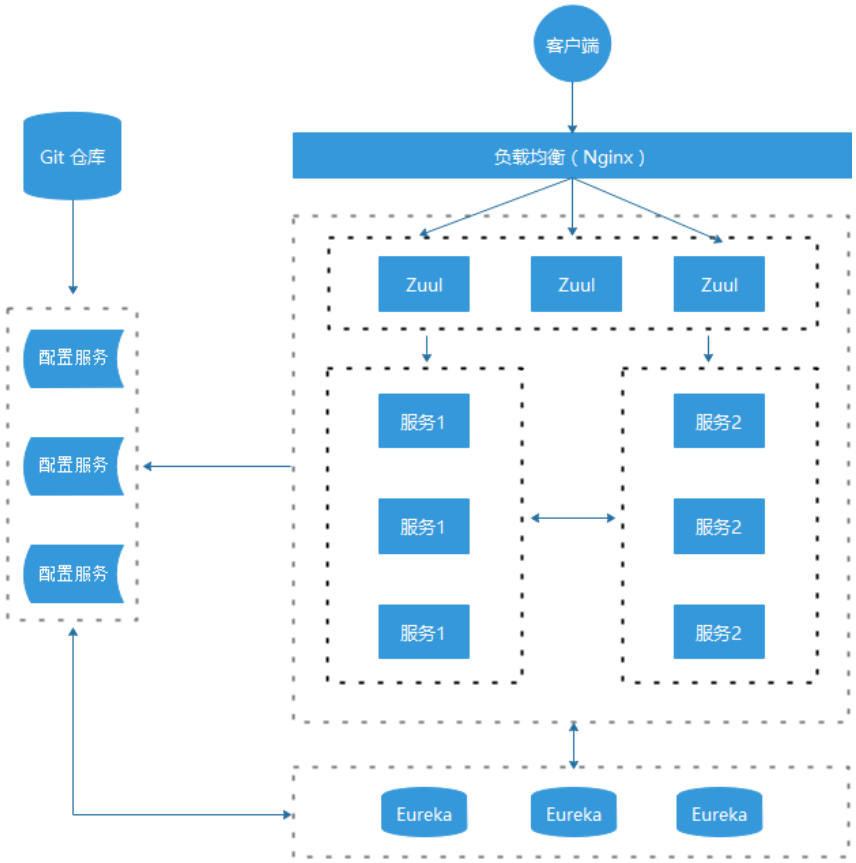


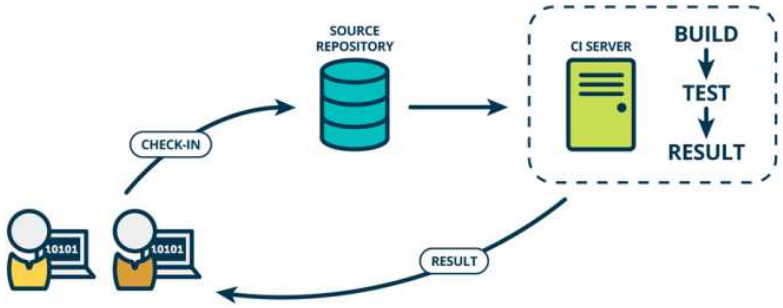


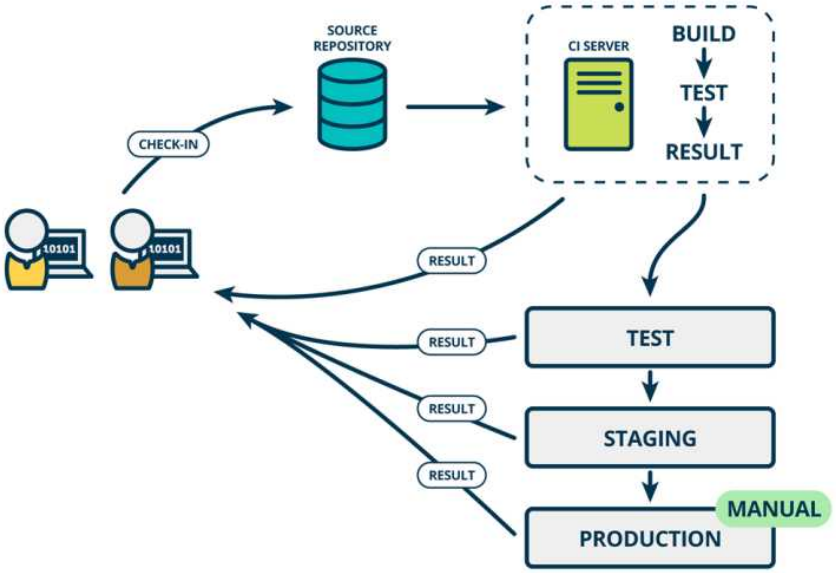
标题	说明
镜像(Images)	Docker 镜像是用于创建 Docker 容器的模板。
容器(Container)	容器是独立运行的一个或一组应用。
客户端(Client)	Docker 客户端通过命令行或者其他工具使用 Docker API (https://docs.docker.com/reference/api/docker_remote_api) 与 Docker 的守护进程通信。
主机(Host)	一个物理或者虚拟的机器用于执行 Docker 守护进程和容器。
仓库(Registry)	Docker 仓库用来保存镜像，可以理解为代码控制中的代码仓库。Docker Hub(https://hub.docker.com) 提供了庞大的镜像集合供使用。
Docker Machine	Docker Machine是一个简化Docker安装的命令行工具，通过一个简单的命令行即可在相应的平台上安装Docker，比如VirtualBox、 Digital Ocean、Microsoft Azure。

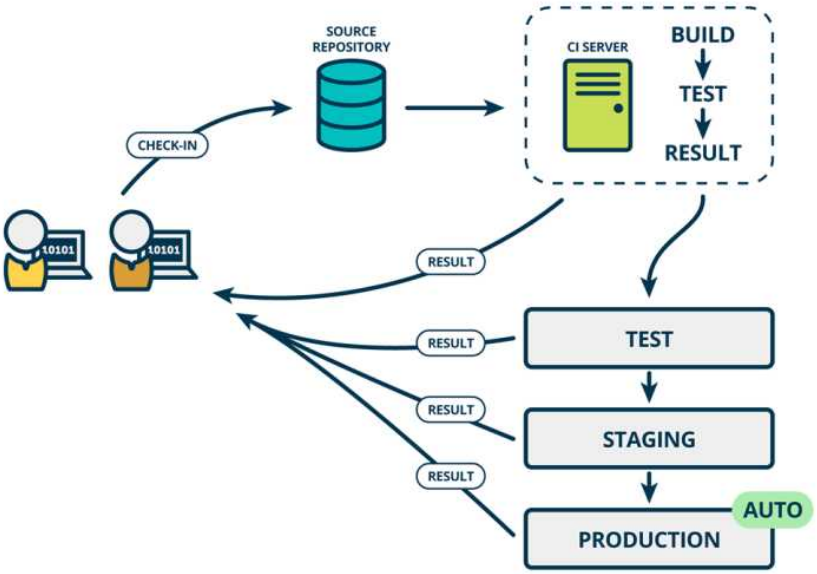


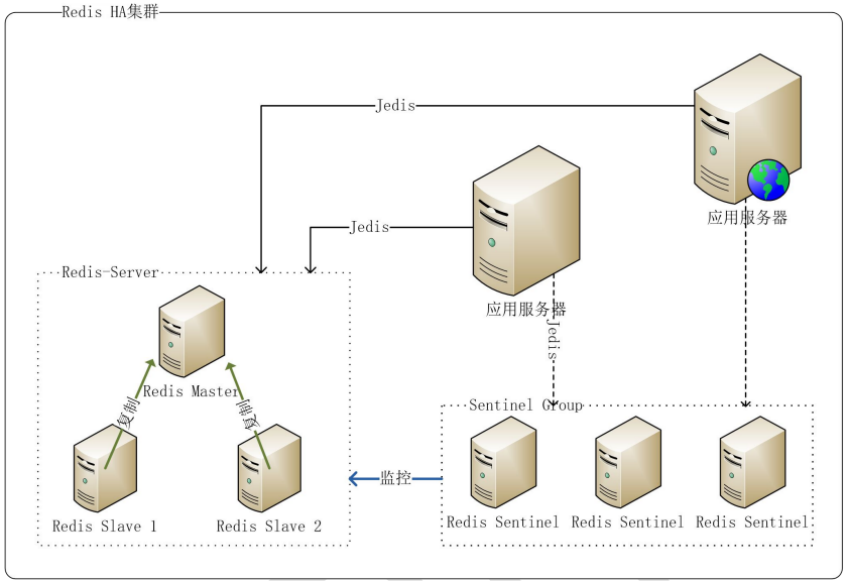

Nginx
Zuul
Docker

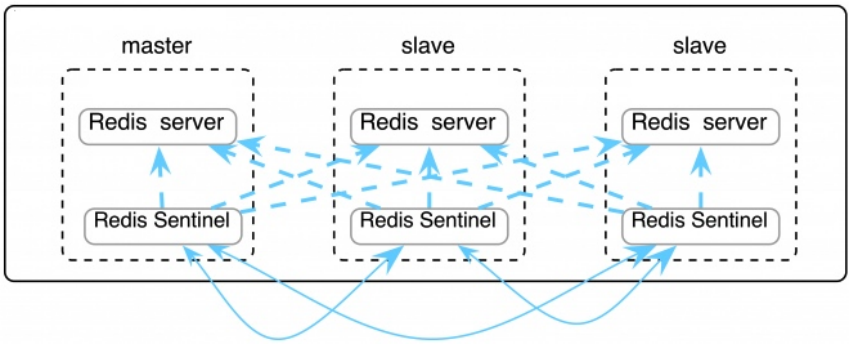

客户端
Git 仓库
负载均衡（Nginx）
Zuul
Zuul
Zuul
配置服务
配置服务
配置服务
服务1
服务1
服务1
服务2
服务2
服务2
Eureka
Eureka
Eureka

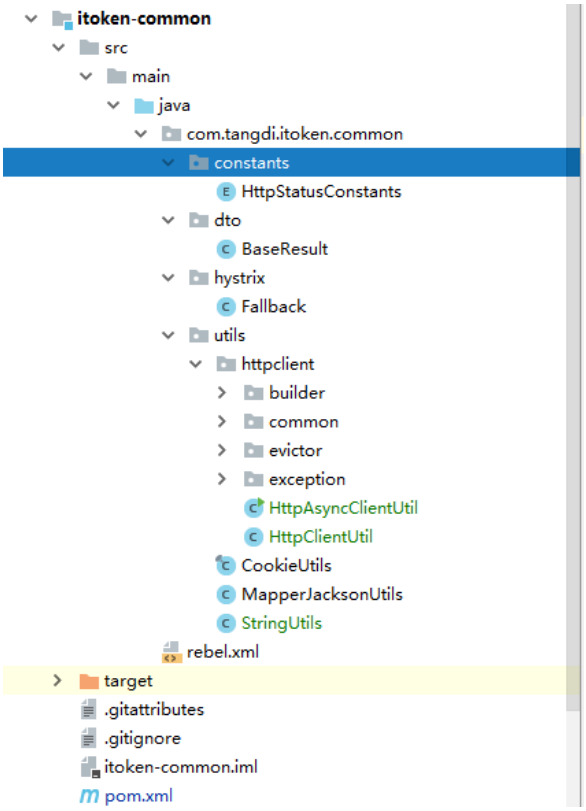

SOURCE REPOSITORY
CI SERVER
BUILD
TEST
RESULT
CHECK-IN
10101
10101
RESULT

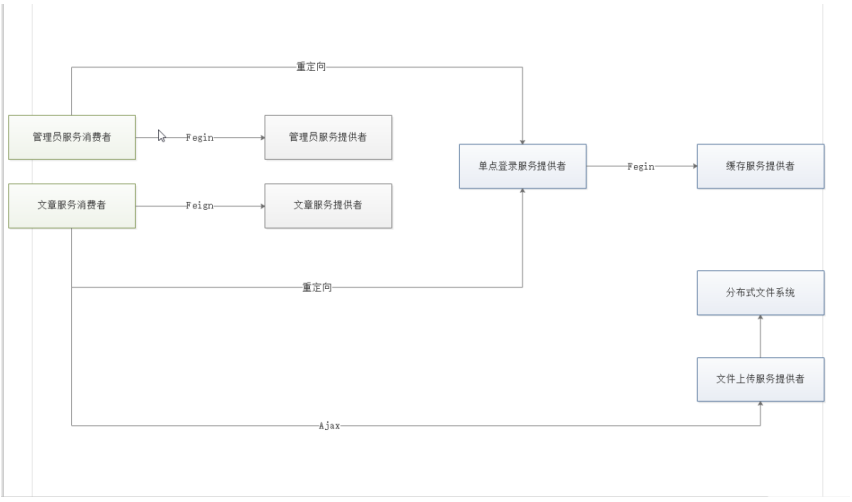

SOURCE REPOSITORY
CI SERVER
BUILD
TEST
RESULT
CHECK-IN
10101
10101
RESULT
RESULT
TEST
RESULT
STAGING
RESULT
PRODUCTION
MANUAL

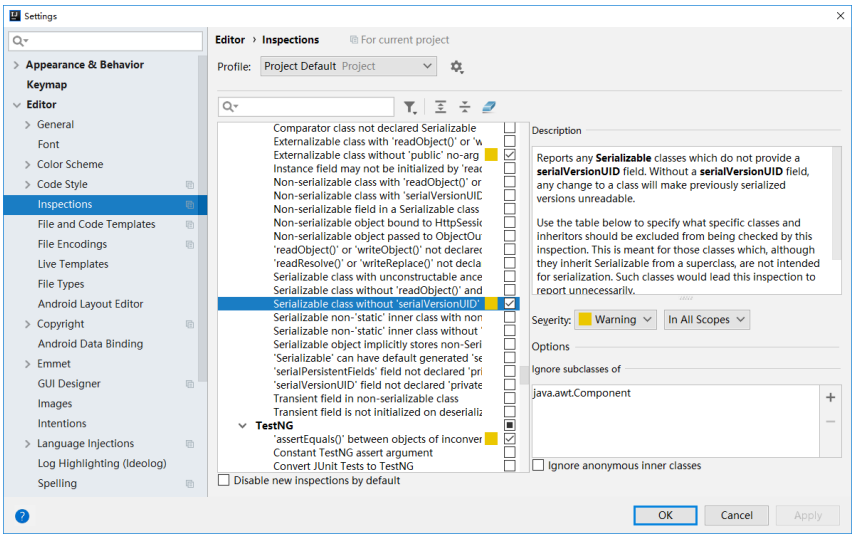

SOURCE REPOSITORY
CI SERVER
BUILD
TEST
RESULT
CHECK-IN
10101
10101
RESULT
RESULT
TEST
RESULT
STAGING
RESULT
PRODUCTION
AUTO

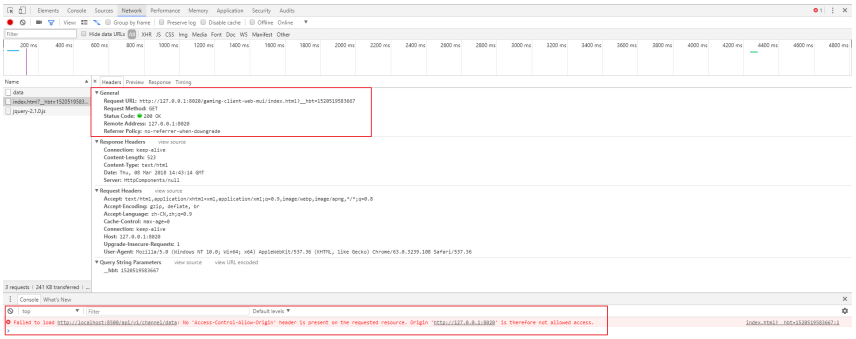

Redis HA集群
Jedis
Jedis
应用服务器
应用服务器
Redis-Server
Redis Master
复制
复制
Jedis
Sentinel Group
监控
Redis Slave 1
Redis Slave 2
Redis Sentinel
Redis Sentinel
Redis Sentinel

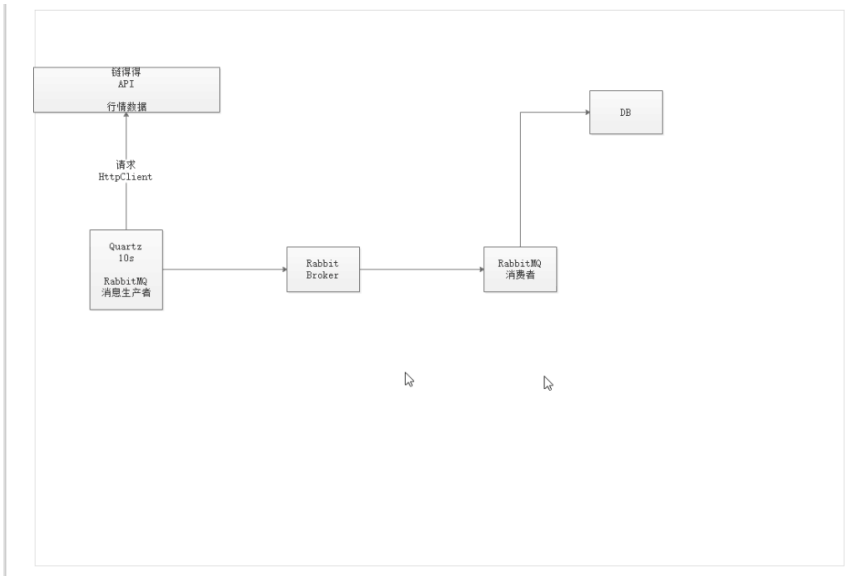

master
slave
slave
Redis server
Redis server
Redis server
Redis Sentinel
Redis Sentinel
Redis Sentinel

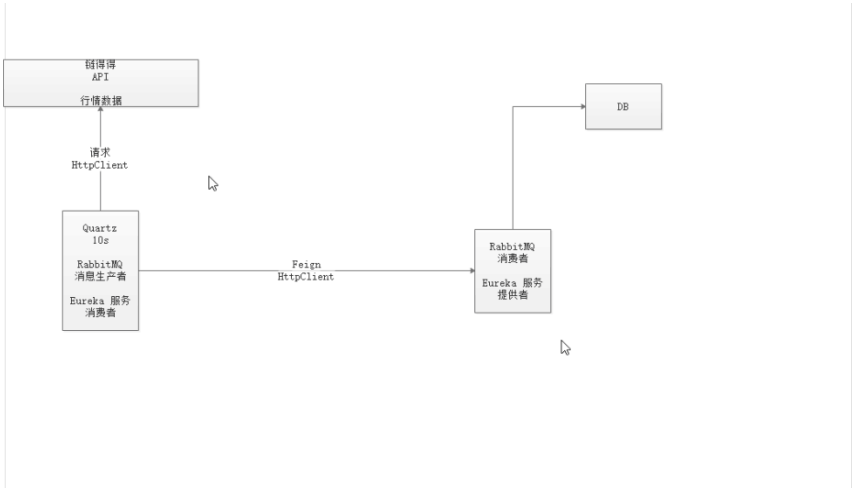

itoken-common
src
main
java
com.tangdi.itoken.common
constants
HttpStatusConstants
dto
BaseResult
hystrix
Fallback
utils
httpclient
builder
common
evictor
exception
HttpAsyncClientUtil
HttpClientUtil
CookieUtils
MapperJacksonUtils
StringUtils
rebel.xml
target
.gitattributes
.gitignore
itoken-common.iml
pom.xml

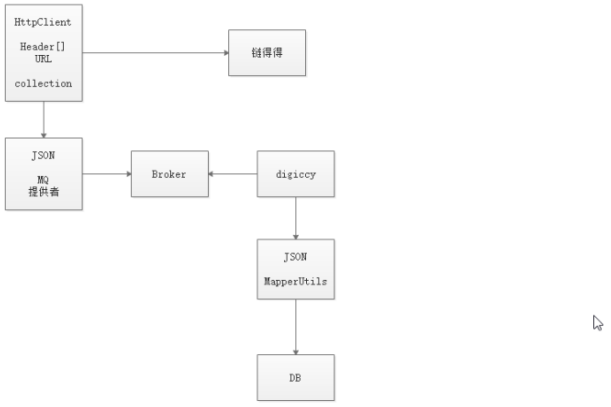

重定向
管理员服务消费者
Fegin
管理员服务提供者
单点登录服务提供者
Fegin
缓存服务提供者
文章服务消费者
Feign
文章服务提供者
重定向
分布式文件系统
文件上传服务提供者
Ajax


Settings
Editor › Inspections
For current project
Profile: Project Default Project
Appearance & Behavior
Keymap
Editor
General
Font
Color Scheme
Code Style
Inspections
File and Code Templates
File Encodings
Live Templates
File Types
Android Layout Editor
Copyright
Android Data Binding
Emmet
GUI Designer
Images
Intentions
Language Injections
Log Highlighting (Ideolog)
Spelling
Comparator class not declared Serializable
Externalizable class with 'readObject()' or 'w
Externalizable class without 'public' no-arg
Instance field may not be initialized by 'read
Non-serializable class with 'readObject()' or
Non-serializable class with 'serialVersionUID
Non-serializable field in a Serializable class
Non-serializable object bound to HttpSessio
Non-serializable object passed to ObjectOu
'readObject()' or 'writeObject()' not declared
'readResolve()' or 'writeReplace()' not decla
Serializable class with unconstructable ance
Serializable class without 'readObject()' and
Serializable class without 'serialVersionUID'
Serializable non-'static' inner class with non
Serializable non-'static' inner class without
Serializable object implicitly stores non-Seri
'Serializable' can have default generated 'se
'serialPersistentFields' field not declared 'pri
'serialVersionUID' field not declared 'private
Transient field in non-serializable class
Transient field is not initialized on deserializ
TestNG
'assertEquals()' between objects of inconver
Constant TestNG assert argument
Convert JUnit Tests to TestNG
Disable new inspections by default
Description
Reports any Serializable classes which do not provide a serialVersionUID field. Without a serialVersionUID field, any change to a class will make previously serialized versions unreadable.
Use the table below to specify what specific classes and inheritors should be excluded from being checked by this inspection. This is meant for those classes which, although they inherit Serializable from a superclass, are not intended for serialization. Such classes would lead this inspection to report unnecessarily.
Severity: Warning
In All Scopes
Options
Ignore subclasses of
java.awt.Component
Ignore anonymous inner classes
OK
Cancel
Apply



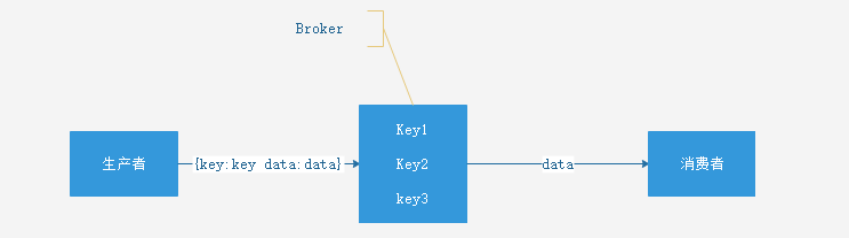

链得得
API
行情数据
请求
HttpClient
Quartz
10s
RabbitMQ
消息生产者
Rabbit
Broker
RabbitMQ
消费者
DB

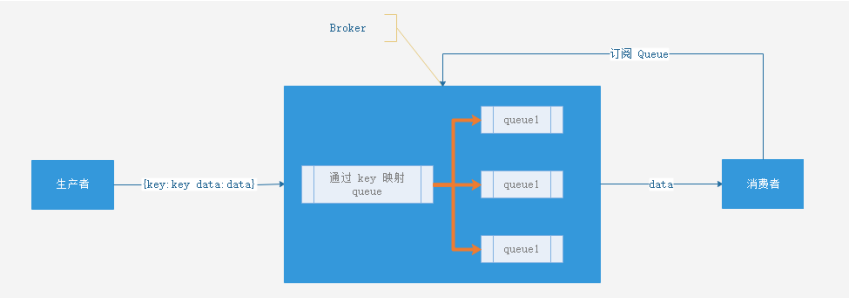


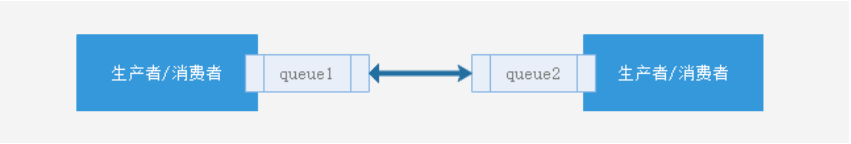


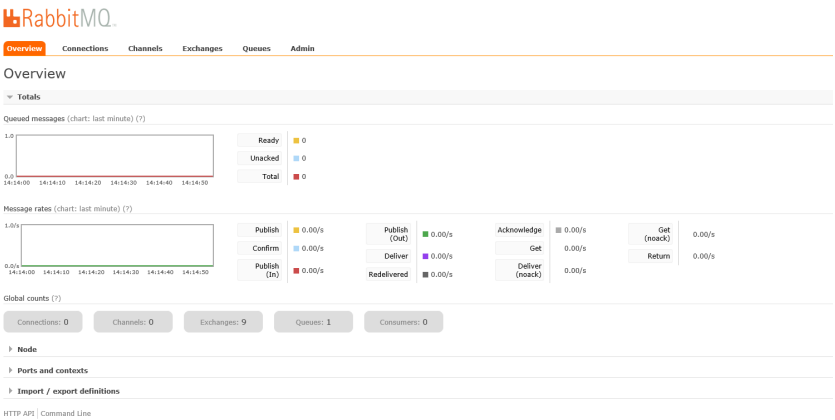

```java
public class ServiceDigiccyApplication  {
    public static void main(String[] args) {
//        SpringApplication.run(ServiceDigiccyApplication.class, args);

        try {
            Header[] headers = {
                new BasicHeader( name: "app-key",  value: "123456789"),
                new BasicHeader( name: "device",  value: "h5"),
                new BasicHeader( name: "identifier",  value: "123456789"),
                new BasicHeader( name: "app-version",  value: "1.0"),
                new BasicHeader( name: "Connection",  value: "Keep-Alive"),
                new BasicHeader( name: "User-Agent",  value: "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/68.0.3440.84 Safari/537.36")
            };

            HttpConfig httpConfig = HttpConfig.custom();
            httpConfig.url("http://ddapi.tmtpost.com/v1/digiccy/exchange");
            httpConfig.headers(headers);

            String result = HttpClientUtil.send(httpConfig);
            System.out.println(result);

            ObjectMapper objectMapper = MapperUtils.getInstance();
            JsonNode jsonNode = objectMapper.readTree(result);
            JsonNode data = jsonNode.findPath("data");
            List<TbDigiccyExchange> list = MapperUtils.json2list(data.toString(), TbDigiccyExchange.class);
            for (TbDigiccyExchange tbDigiccyExchange : list) {
                System.out.println(tbDigiccyExchange.getExchName());
            }
        } catch (HttpProcessException e) {
```

```java
    private String exchGuid;

    /**
     * 交易所名称
     */
    @JsonProperty(value = "exch_name")
    @Column(name = "exch_name")
    private String exchName;

    /**
     * 交易所代号
     */
    @JsonProperty(value = "exch_code")
    @Column(name = "exch_code")
    private String exchCode;

    /**
     * 交易所网址
     */
    @JsonProperty(value = "exch_url")
    @Column(name = "exch_url")
    private String exchUrl;

    /**
     * 获取交易所编码
     *
     * @return exch_guid - 交易所编码
     */
    public String getExchGuid() { return exchGuid; }

    /**
```



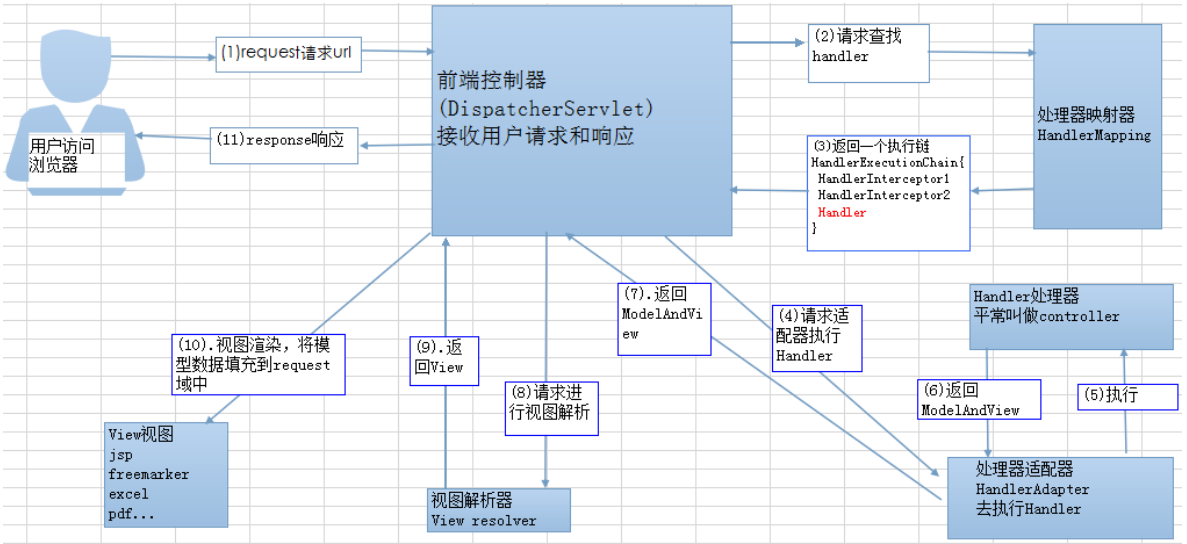




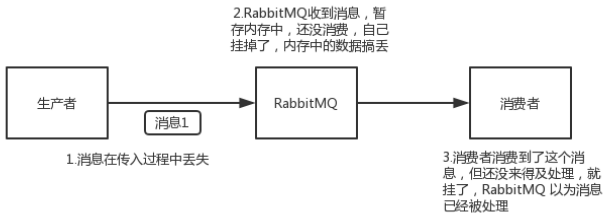


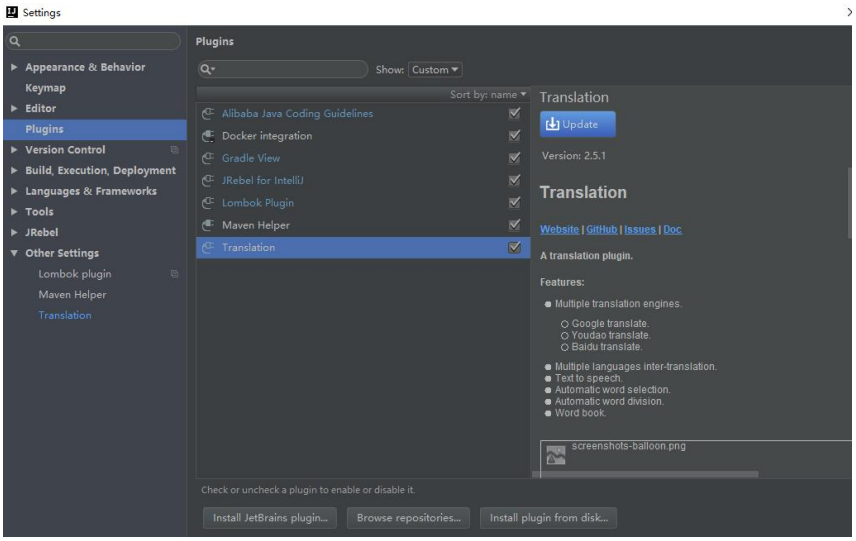
字段	允许值	允许的特殊字符
秒	0-59	, - * /
分	0-59	, - * /
小时	0-23	, - * /
日期	1-31	, - * / L W C
月份	1-12 或者 JAN-DEC	, - * /
星期	1-7 或者 SUN-SAT	, - * / L C #
年（可选）	留空, 1970-2099	, - * /

表达式	意义
0 0 12 * * ?	每天中午 12 点触发
0 15 10 ? * *	每天上午 10:15 触发
0 15 10 * * ?	每天上午 10:15 触发
0 15 10 * * ? *	每天上午 10:15 触发
0 15 10 * * ? 2005	2005 年的每天上午 10:15 触发
0 * 14 * * ?	在每天下午 2 点到下午 2:59 期间的每 1 分钟触发
0 0/5 14 * * ?	在每天下午 2 点到下午 2:55 期间的每 5 分钟触发
0 0/5 14,18 * * ?	在每天下午 2 点到 2:55 期间和下午 6 点到 6:55 期间的每 5 分钟触发
0 0-5 14 * * ?	在每天下午 2 点到下午 2:05 期间的每 1 分钟触发
0 10,44 14 ? 3 WED	每年三月的星期三的下午 2:10 和 2:44 触发
0 15 10 ? * MON-FRI	周一至周五的上午 10:15 触发
0 15 10 15 * ?	每月 15 日上午 10:15 触发
0 15 10 L * ?	每月最后一日的上午 10:15 触发
0 15 10 ? * 6L	每月的最后一个星期五上午 10:15 触发
0 15 10 ? * 6L 2002-2005	2002 年至 2005 年的每月的最后一个星期五上午 10:15 触发
0 15 10 ? * 6#3	每月的第三个星期五上午 10:15 触发

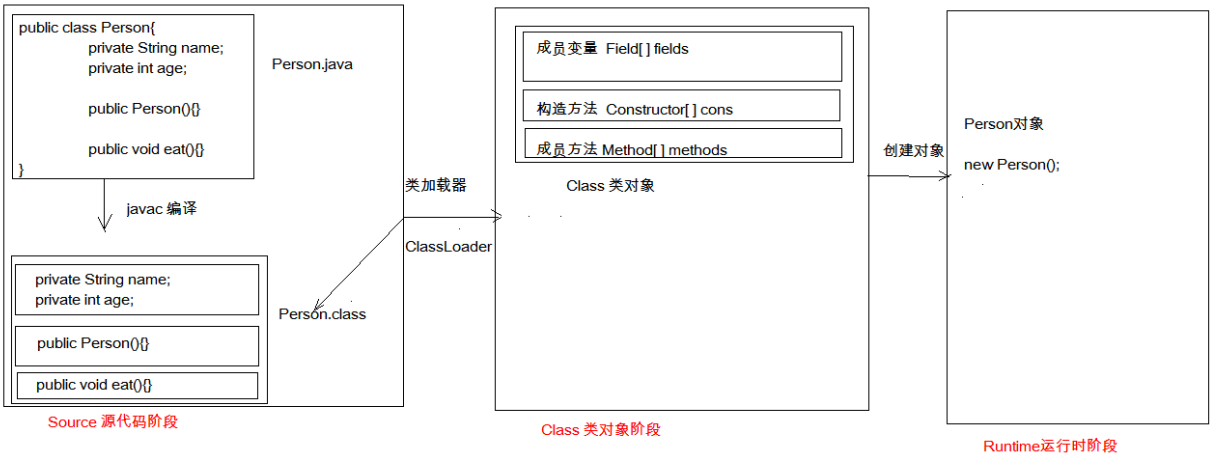


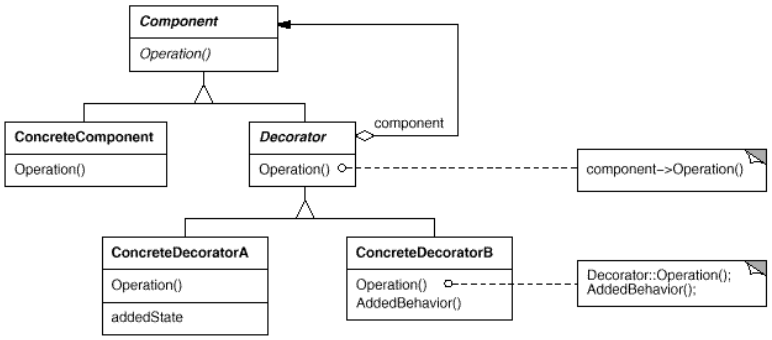
RabbitMQ 消息丢失的 3 种情况

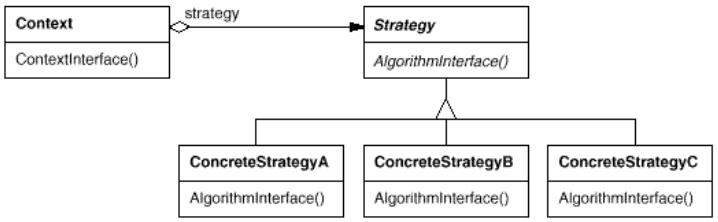


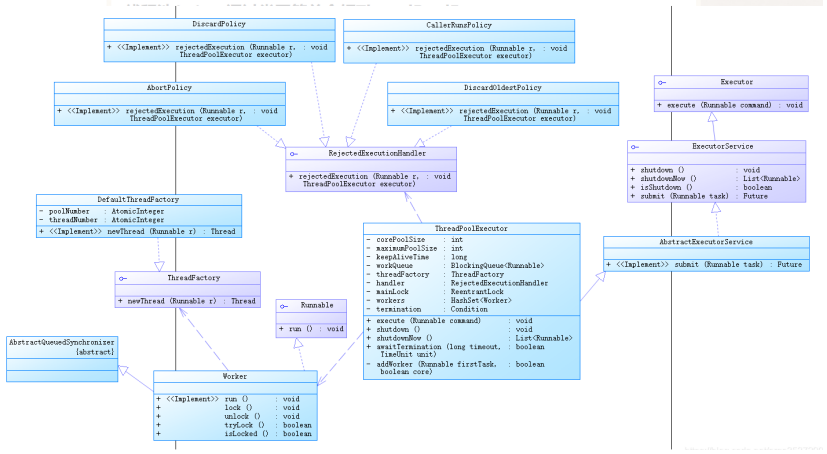


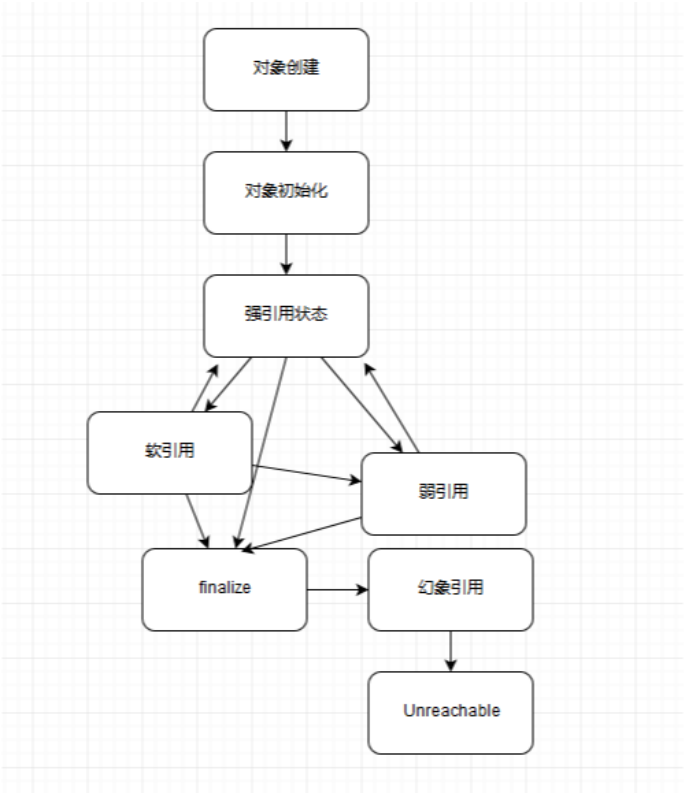
Java代码 在计算机中 经历的阶段：三个阶段












```
    PreparedStatement preparedStatement = null;
    ResultSet resultSet = null;
try {
    // 加载数据库驱动
    Class.forName("com.mysql.jdbc.Driver");
    // 通过驱动管理类获取数据库链接
    connection =
DriverManager.getConnection("jdbc:mysql://localhost:3306/mybatis?
characterEncoding=utf-8", "root", "root");
    // 定义sql语句 ?表示占位符
    String sql = "select * from user where username = ?";
    // 获取预处理statement
    preparedStatement = connection.prepareStatement(sql);
    // 设置参数，第一个参数为sql语句中参数的序号（从1开始），第二个参数为设置的参数值
    preparedStatement.setString(1, "tom");
    // 向数据库发出sql执行查询，查询出结果集
    resultSet = preparedStatement.executeQuery();
    // 遍历查询结果集
    while (resultSet.next()) {
        int id = resultSet.getInt("id");
        String username = resultSet.getString("username");
    // 封装User
        user.setId(id);
        user.setUsername(username);
    }
    System.out.println(user);

    }
```

JDBC问题分析：

1. 数据库配置信息存在硬编码问题
2. 频繁创建释放数据库连接

解决：1.配置文件
2. 连接池

1.sql语句、设置参数、获取结果集参数均存在硬编码问题 解决：配置文件

1.手动封装返回结果集，较为繁琐 解决：反射、内省

自定义持久层框架设计思路：

自定义持久层框架设计思路：

使用端：（项目）：引入自定义持久层框架的jar包
提供两部分配置信息：数据库配置信息、sql配置信息：sql语句、参数类型、返回值类型
使用配置文件来提供这两部分配置信息：
（1）sqlMapConfig.xml：存放数据库配置信息，存放mapper.xml的全路径
（2）mapper.xml：存放sql配置信息

自定义持久层框架本身：（工程）：本质就是对JDBC代码进行了封装
（1）加载配置文件：根据配置文件的路径，加载配置文件成字节输入流，存储在内存中
创建Resources类　方法：InputSteam getResourceAsSteam(String path)

（2）创建两个javaBean：（容器对象）：存放的就是对配置文件解析出来的内容
Configuration：核心配置类：存放sqlMapConfig.xml解析出来的内容
MappedStatement：映射配置类：存放mapper.xml解析出来的内容

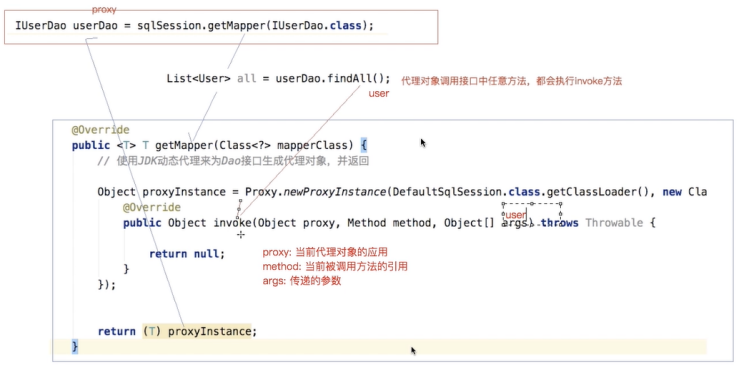
（3）解析配置文件：dom4j
创建类：SqlSessionFactoryBuilder　方法：build(InputSteam in)
第一：使用dom4j解析配置文件，将解析出来的内容封装到容器对象中
第二：创建SqlSessionFactory对象；生产sqlSession :会话对象（工厂模式）

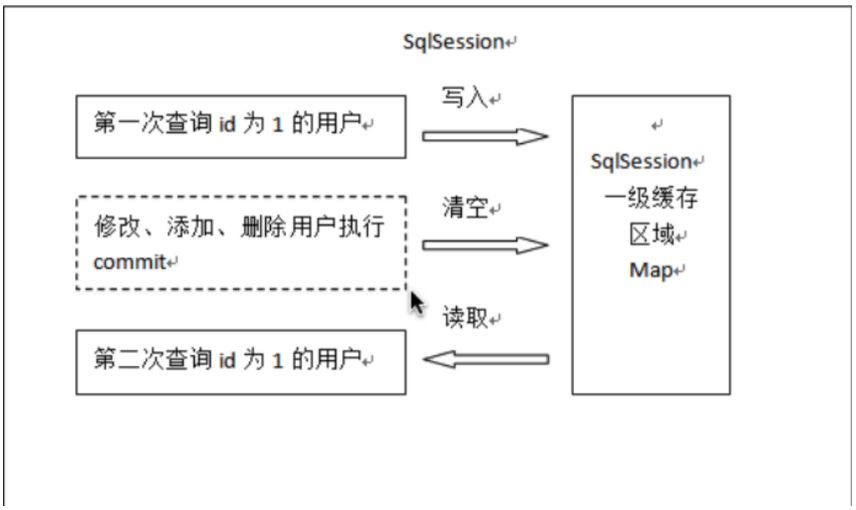
（4）创建SqlSessionFactory接口及实现类DefaultSqlSessionFacotry
第一：openSession(): 生产sqlSession

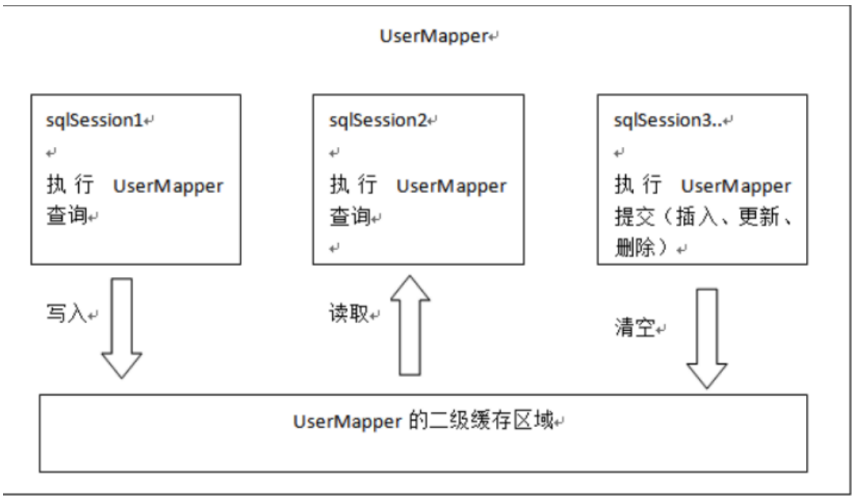
（5）创建SqlSession接口及实现类DefaultSession
定义对数据库的crud操作：selectList()
selectOne()
update()z
delete()

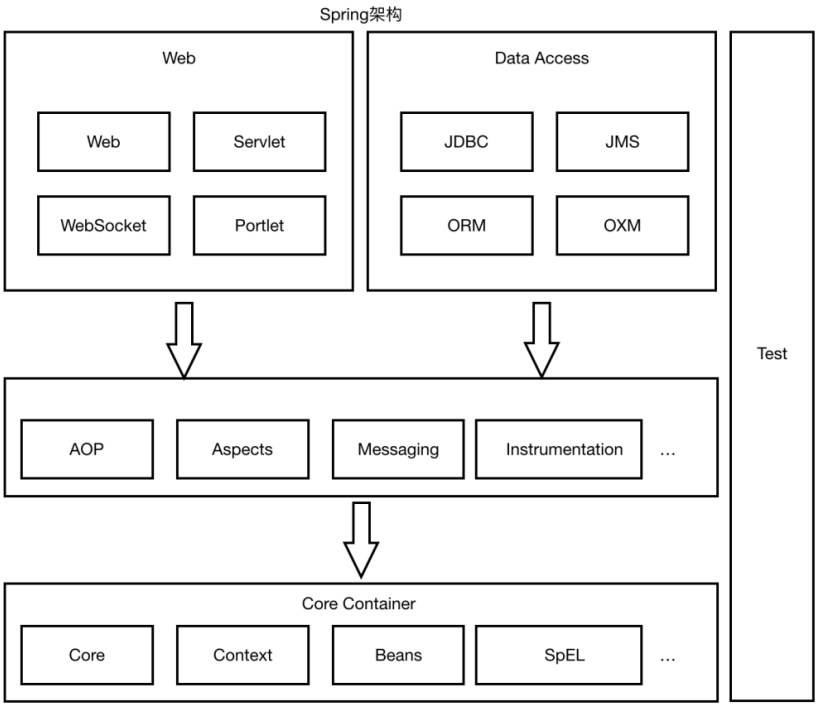
（6）创建Executor接口及实现类SimpleExecutor实现类
query(Configuration,MappedStatement,Object... params): 执行的就是JDBC代码

解决思路：使用代理模式生成Dao层接口的代理实现类






UserMapper
sqlSession1
执行 UserMapper 查询
sqlSession2
执行 UserMapper 查询
sqlSession3..
执行 UserMapper 提交（插入、更新、删除）
写入
读取
清空
UserMapper 的二级缓存区域


Spring架构
Web
Web
Servlet
WebSocket
Portlet
Data Access
JDBC
JMS
ORM
OXM
Test
AOP
Aspects
Messaging
Instrumentation
...
Core Container
Core
Context
Beans
SpEL
...