**网络思想意识形态分析系统设计说明书**

1. **引言**

## 目的

网络所具有的空间性、社会性、文化性、政治性等特征表明,网络是现实社会的延伸。网络意识形态不是别的,是现实国家、社会与执政党的意识形态在网络空间的呈现。网络意识形态工作必须以占领网络阵地、保证网络空间和网络社会的主权为己任,不断创新网络党委、网络行政和网络文化建设,完善网络技术和网络法律,进行网络治理,才能牢牢掌握网络意识形态的领导权和管理权。加强互联网内容建设，建立网络综合治理体系，营造清朗的网络空间刻不容缓。

## **背景**

**网络意识形态领域的社会治理能力显著提升**。加强互联网内容建设，建立网络综合治理体系，营造清朗的网络空间，是党的十九大报告对于网络空间意识形态工作领导权的明确阐述。2018年，中央网络安全和信息化委员会成立、《微博客信息服务管理规定》实施、《具有舆论属性或社会动员能力的互联网信息服务安全评估规定》施行、互联网法院挂牌和互联网联合辟谣平台上线，这一系列举措有效地制止了危害国家安全和社会稳定、侵害他人合法权益、误导社会舆论等有害信息在网络自媒体上的泛滥。10月，国家网信办会同有关部门专项整治自媒体领域的乱象，依法依规处置9800多个自媒体账号。北京、河北、湖南、湖北、广东等多省市开展专项整治行动，根据网民举报和自主监管及时清理各类不良信息，处置属地违法违规互联网平台，令违法乱纪、蛊惑人心的不良信息无处藏身。同时，互联网信息平台企业对于即时通信产品内容监管的主体责任和意识进一步增强，自我监管意识和自律能力进一步提升。以微信为例，2018年2月起，微信社交平台先后对具有用户诱导行为的春节活动、未取得信息网络传播视听节目许可的短视频链接、存在谩骂和地域歧视等不文明行为的“对骂群”等违规内容采取处理措施。除了微信以外，微博、今日头条、抖音等都加大了对于违规帐号的处罚力度，处罚包括扣分、禁言以及永久封禁。由此可见，党委领导、政府管理、企业履责、社会监督、网民自律等多主体参与，经济、法律、技术等多种手段相结合的综合治网格局建设已经初见成效，网络社交平台的思想舆论生态持续改善。

**网络空间的社会共识显著增强。**构建网上网下同心圆，更好凝聚社会共识，巩固全党全国人民团结奋斗的共同思想基础，是网络意识形态建设的重要内容。2018年，人民网、新华网等多家网络媒体围绕庆祝改革开放四十周年主题展开的系列宣传活动受到了网络广泛响应与好评。爱国主义题材电影《红海行动》继2017年《战狼2》之后，在网上受到广泛好评。《中华人民共和国英雄烈士保护法》得到全国人大常委会全票表决通过，受到网民的广泛支持。与上述支持态度形成鲜明对比的是，精日分子穿军国主义服装在抗战纪念馆拍照、D&G辱华言论事件遭到网民一致声讨，明星逃税事件、高铁霸座事件和网约车乘客安全事故受到网民一致谴责。这一系列事件中呈现出来的网络民意，突显了网民强烈的国家民族意识和爱憎分明的道德自觉，表明了社会基本道德领域中的共识程度不断加深。

**传统企业与网络平台的媒体化发展趋势更加明显。**2018年，传统企业和互联网平台纷纷突破某种单一业务的运营模式，在跨界融合经营中强化了思想社交功能。时下，绝大多数中国企业都把微博、微信作为企业宣传的标准配置，通过增强在互联网上的影响力，以实现主营产品和服务的销售目的。互联网平台企业的跨界融合速度显著加快，思想交流功能不再局限于专业性的新闻与社交媒体之中。网络购物、旅行预订、网上支付、在线教育、网络约车和互联网理财等网络应用行业均出现了跨界融合趋向，平台企业围绕核心业务进行增值化服务、全面化布局，不断由单一业务向平台化生态拓展，议题设置、议程引导和多点互动的社交功能成为互联网口碑经济模式的基础配置。不仅如此，纸媒、电台、电视台等传统媒体也在主动入驻微博、微信等网络自媒体平台，通过吸引粉丝以增加用户数量。各级党政机关和群团组织等积极运用微博、微信、客户端等“两微一端”新媒体，发布政务信息、回应社会关切、推动社会协同治理，不断提升地方政府信息公开化、服务线上化水平。当前，具有社交链接属性和资讯聚合属性的各类网络平台媒体不断探索“内容+渠道+搜索+大数据”的全新媒体运营模式，已然成为社会信息集成、流动、分发和传播的中枢，推动了社会思想舆论格局的进一步分化重组。

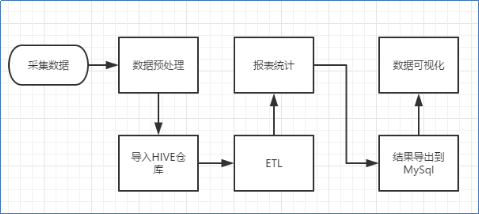
# 总体设计

1. 需求概述

网络思想意识形态分析系统主要包括以下功能：

1. 关注广大网民在社交媒体上发表的言论；
2. 关注广大网民浏览的网站；
3. 识别诈骗网站；
4. 统计广大网民关注的问题TOP5；
5. 统计广大网民关注论坛TOP5;
6. **业务架构**

如下图所示，将采集到的数据进行数据预处理后导入HIVE仓库，在经过ETL对数据进行处理进行报表统计，把统计结果导出到MySQL数据库后通过接口对数据可视化展示。

****

业务流程图

大数据架构中，除了hdfs+mapreduce+hive组成分析系统的核心之外，还需要数据采集、结果数据导出、任务调度等不可或缺的辅助系统。如下图所示：

1. 数据采集：定制开发采集程序，使用开源框架FLUME

package com.yc.flume;

import java.util.List;

import java.util.Map;

import org.apache.flume.Context;

import org.apache.flume.Event;

import org.apache.flume.interceptor.Interceptor;

import com.google.common.collect.Lists;

public class MyInterceptor implements Interceptor{

public void close() {

}

public void initialize() {

}

public Event intercept(Event event) {

Map<String, String> map = event.getHeaders();

String path = map.get("filePath");

String [] arr = path.split("/");

String newPath = new StringBuilder().append("/").append(arr[6]).append("/").append(arr[7]).append("/").append(arr[8].substring(0, arr[8].indexOf("."))).toString();

map.put("filePath", newPath);

event.setHeaders(map);

return event;

}

public List<Event> intercept(List<Event> events) {

List<Event> intercepted = Lists.newArrayListWithCapacity(events.size());

for (Event event : events) {

Event interceptedEvent = intercept(event);

if (interceptedEvent != null) {

intercepted.add(interceptedEvent);

}

}

return intercepted;

}

public static class Builder implements Interceptor.Builder{

public void configure(Context arg0) {

}

public Interceptor build() {

return new MyInterceptor();

}

}

}

通过FLUME对数据进行采集、清洗和转换。

　　　　2) 数据预处理：定制开发mapreduce程序运行于hadoop集群

　　　　3) 数据仓库技术：基于hadoop之上的Hive

package com.yc.hive;

import org.apache.hadoop.hive.ql.exec.UDF;

import org.jdom.Document;

import org.jdom.Element;

import org.jdom.JDOMException;

import org.jdom.Text;

import org.jdom.input.SAXBuilder;

import java.io.ByteArrayInputStream;

import java.io.IOException;

import java.io.InputStream;

import java.nio.charset.StandardCharsets;

import java.util.HashMap;

import java.util.List;

import java.util.Map;

import java.util.Set;

public class ResolveXml extends UDF {

public static String evaluate(String strxml) {

// if ("".equals(strxml)) {

// return null;

// }

Map<String, String> m = new HashMap<String, String>();

try {

InputStream in = new ByteArrayInputStream(strxml.getBytes(StandardCharsets.UTF\_8));

SAXBuilder builder = new SAXBuilder();

Document doc = builder.build(in);

// 解析 DOM

Element root = doc.getRootElement();

List list = root.getChildren();

for (Object o : list) {

Element e = (Element) o;

if (e.getContentSize() == 0)

continue;

Text t = (Text) e.getContent().get(0);

m.put(e.getAttribute("n").getValue(), t.getValue());

}

// 关闭流

in.close();

} catch (Exception e) {

e.printStackTrace();

// throw new IOException(e.getMessage()); // 统一转化为 IO 异常输出

}

Set<String> set = m.keySet();

String res = null;

for (String string : set) {

if (string.equals("trace\_t")) {

String result = m.get(string);

if (result.equals("web\_url")) {

for (String string2 : set) {

if (string2.equals("url")) {

res = m.get(string2);

}

}

}

}

}

return res;

}

}

使用HIVEUDF函数对数据进行统计。

1. 数据导出：基于hadoop的sqoop数据导入导出工具

sqoop import \

--connect jdbc:mysql://hadoop102:3306/test \

--username root \

--password root \

--table student \

--columns sid,sname \

--hive-import \

--fields-terminated-by '|' \

--hive-table test.mate \

--m 1

通过sqoop组件将统计过的数据导入MySQL数据库中。

　　　　5) 数据可视化：定制开发web程序

　　　　6) 整个过程的流程调度：hadoop生态圈中的oozie工具

nameNode=hdfs://hadoop102:9000

jobTracker=hadoop102:8032

queueName=default

launcher\_queName=oozie

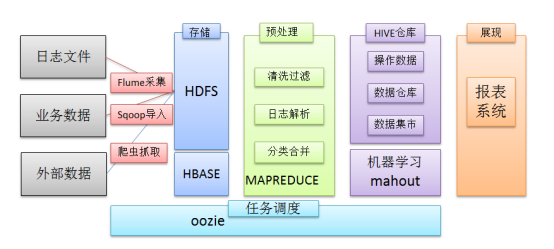
jobName=hive\_test

workflowUri=${nameNode}/user/hadoop/oozie-apps/hive

oozie.use.system.libpath=true

oozie.wf.application.path=${workflowUri}

利用Oozie对任务进行定时调度。



大数据架构图

# 系统功能模块实现

## Web前端页面通过ECharts来展示

ECharts，一个使用 JavaScript 实现的开源可视化库，可以流畅的运行在 PC 和移动设备上，兼容当前绝大部分浏览器（IE8/9/10/11，Chrome，Firefox，Safari等），底层依赖矢量图形库 [ZRender](https://github.com/ecomfe/zrender" \t "https://www.echartsjs.com/zh/_blank)，提供直观，交互丰富，可高度个性化定制的数据可视化图表。ECharts 提供了常规的[折线图](https://www.echartsjs.com/zh/option.html" \l "series-line" \t "https://www.echartsjs.com/zh/_blank)、[柱状图](https://www.echartsjs.com/zh/option.html" \l "series-bar" \t "https://www.echartsjs.com/zh/_blank)、[散点图](https://www.echartsjs.com/zh/option.html" \l "series-scatter" \t "https://www.echartsjs.com/zh/_blank)、[饼图](https://www.echartsjs.com/zh/option.html" \l "series-pie" \t "https://www.echartsjs.com/zh/_blank)、[K线图](https://www.echartsjs.com/zh/option.html" \l "series-candlestick" \t "https://www.echartsjs.com/zh/_blank)，用于统计的[盒形图](https://www.echartsjs.com/zh/option.html" \l "series-boxplot" \t "https://www.echartsjs.com/zh/_blank)，用于地理数据可视化的[地图](https://www.echartsjs.com/zh/option.html" \l "series-map" \t "https://www.echartsjs.com/zh/_blank)、[热力图](https://www.echartsjs.com/zh/option.html" \l "series-heatmap" \t "https://www.echartsjs.com/zh/_blank)、[线图](https://www.echartsjs.com/zh/option.html" \l "series-lines" \t "https://www.echartsjs.com/zh/_blank)，用于关系数据可视化的[关系图](https://www.echartsjs.com/zh/option.html" \l "series-graph" \t "https://www.echartsjs.com/zh/_blank)、[treemap](https://www.echartsjs.com/zh/option.html" \l "series-treemap" \t "https://www.echartsjs.com/zh/_blank)、[旭日图](https://www.echartsjs.com/zh/option.html" \l "series-sunburst)，多维数据可视化的[平行坐标](https://www.echartsjs.com/zh/option.html" \l "series-parallel" \t "https://www.echartsjs.com/zh/_blank)，还有用于 BI 的[漏斗图](https://www.echartsjs.com/zh/option.html" \l "series-funnel" \t "https://www.echartsjs.com/zh/_blank)，[仪表盘](https://www.echartsjs.com/zh/option.html" \l "series-gauge" \t "https://www.echartsjs.com/zh/_blank)，并且支持图与图之间的混搭。除了已经内置的包含了丰富功能的图表，ECharts 还提供了[自定义系列](https://www.echartsjs.com/zh/option.html" \l "series-custom)，只需要传入一个renderItem函数，就可以从数据映射到任何你想要的图形，更棒的是这些都还能和已有的交互组件结合使用而不需要操心其它事情。

ECharts 支持以 Canvas、SVG（4.0+）、VML 的形式渲染图表。VML 可以兼容低版本 IE，SVG 使得移动端不再为内存担忧，Canvas 可以轻松应对大数据量和特效的展现。不同的渲染方式提供了更多选择，使得 ECharts 在各种场景下都有更好的表现。

<script type="text/javascript">

window.onload = function (){

$.ajax({

url:'',

type:'post',

datatype:'json',

success : function(data){

var partner = new Array();

for(var i=0;i<data.length;i++){

partner.push(data[i].partner);

}

var odata=[];

for(var i=0;i<data.length;i++){

var obj={};

obj.name=partner[i];

obj.type='bar';

obj.data=[data[i].qyearOne,data[i].qyearTwo,data[i].qyearThree,data[i].qyearFour,data[i].yearOne,data[i].yearTwo,data[i].yearThree,data[i].yearFour,data[i].hyearOne,data[i].hyearTwo,data[i].hyearThree,data[i].hyearFour];

obj.barWidth=30;//宽度

odata.push(obj);

}

option = {

tooltip : {

trigger: 'axis'

},

legend: {

data:partner

},

toolbox: {

show : true,

},

calculable : true,

xAxis : [{

type : 'category',

data : [qyear+'Q1',qyear+'Q2',qyear+'Q3',qyear+'Q4',year+'Q1',year+'Q2',year+'Q3',year+'Q4',hyear+'Q1',hyear+'Q2',hyear+'Q3',hyear+'Q4']

}],

yAxis : [{

type : 'value'

}],

series : odata //前台组装数据

};

//获取要赋值的DOM控件

var myChart = echarts.init(document.getElementById('chartmain'));

//赋值

myChart.setOption(option);

}

});

</script>

## 引入 ECharts 图表和组件

// 引入 ECharts 主模块

var echarts = require('echarts/lib/echarts');

// 引入柱状图require('echarts/lib/chart/bar');

// 引入提示框和标题组件require('echarts/lib/component/tooltip');

require('echarts/lib/component/title');

// 基于准备好的dom，初始化echarts实例

var myChart = echarts.init(document.getElementById('main'));

// 绘制图表

myChart.setOption({

title: { text: 'ECharts' },

tooltip: {},

xAxis: {

data: ["百度论坛","天涯论坛","猫扑社区","凤凰论坛","搜狐论坛",]

},

yAxis: {},

series: [{

name: '排行',

type: 'bar',

data: [5, 20, 36, 10, 20]

}]});

## 后端数据接口采用微服务架构

微服务架构是一种将单应用程序作为一套小型服务开发的方法，每种应用程序都在其自己的进程中运行，并与轻量级机制（通常是HTTP资源的API）进行通信。这些服务是围绕业务功能构建的，可以通过全自动部署机制进行独立部署。这些服务的集中化管理已经是最少的，它们可以用不同的编程语言编写，并使用不同的数据存储技术。在开始介绍微服务风格（microservice style）前，比较一下整体风格（monolithic style）是很有帮助的：一个完整应用程序（monolithic application）构建成一个单独的单元。企业应用程序通常建立在三个主要部分中：一个客户端用户界面（由用户计算机上的浏览器中运行的HTML页面和JavaScript组成）数据库（包括插入常见的通常是关系数据库管理的多个表系统）和一个服务器端应用程序。服务器端应用程序将处理HTTP请求，执行特定领域逻辑，通过数据库进行检索和更新数据，选择并填充要发送到浏览器的HTML视图。这个服务器端应用程序是一个庞然大物 - 一个逻辑可执行文件[2]。系统的任何更改都涉及构建和部署新版本的服务器端应用程序。

这样的整体服务（monolithic server）是一种构建系统很自然的方式。处理请求的所有逻辑都在一个进程中运行，允许您使用语言的基本功能将应用程序划分为类，函数和名称空间。谨慎操作时，您可以在开发人员的笔记本电脑上运行和测试应用程序，并使用部署通道来确保更改经过适当测试并部署到生产环境中。您可以通过在负载平衡器后面运行多个实例来横向缩放整体。

单体式应用程序可以取得成功，但越来越多的人会感到失望 - 尤其是随着更多应用程序被部署到云中。变更周期是连在一起的 - 对应用程序的一小部分进行更改，需要重建和部署整个程序。随着时间的推移，它通常很难保持良好的模块化结构，使得难以保持应该：模块内的一个改动仅影响该模块本身中。自适应需要自适应整个应用程序，而不是它的一部分，这样做需要更多资源。

这些挫折引出了微服务架构风格：将应用程序构建为服务套件。除了服务是可独立部署和可伸缩的事实之外，每个服务还提供了一个严格的模块边界，甚至允许用不同的编程语言编写不同的服务。它们也可以由不同的团队来管理。

我们并不是说微服务风格是新颖的或创新的，它的根源至少可以追溯到Unix的设计原则。但我们确实认为，没有足够多的人考虑使用微服务架构，如果他们使用了，那么许多软件开发将会更好。

**微服务架构的九大特性：**

1)服务组件化；

2)按业务组织团队；

3)做”产品“的态度；

4)智能端点与哑管道（服务调用方式，实时，异步中间件）

5)去中心化治理（组件能针对不同的业务特点选择不同的技术平台）

6)去中心化管理数据（多个不同的MySql实例，各服务之间进行“无事务”的调用，数据一致性，只要求数据在最后的处理状态是一致的即可。补偿机制）

7)基础设施自动化（自动化测试、自动化部署）

8)容错设计（快速检测出故障资源并尽可能地自动回复服务是必须被设计和考虑的）

9)演进式设计

**微服务优点：**

## 技术异构性

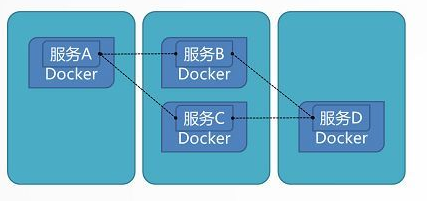
在单体架构下,会非常依赖于项目一开始对技术的选择,一旦选择了个技术栈,之后几年都会被绑定在这样个技术栈下,很难应对变化。给我们提供了一个更细粒度使用技术的可能在不同的服务里可以使用完全不同的技术栈不同的语言、框架甚至数据库,真正做到用最适合的技术解决最适合的问题,从而让我们可以更加敏捷地响应需求和市场的变化提高了竞争力。

## 弹性

弹性弹性工程学的一个关键概念是舱壁。如果系统中的一个组件不可用了,但并没有导致级联故障,那么系统的其他部分还可以正常运行。服务边界就是一个很显然的舱壁。在单块系统中,如果服务不可用,那么所有的功能都会不可用。对于单块服务的系统而言,可以通过将同样的实例运行在不同的机器上来降低功能完全不可用的概率,然而微服务系统本身就能够很好地处理服务不可用和功能降级问题。

## 扩展

庞大的单块服务只能作为一个整体进行扩展即使系统中只有一小部分存在性能问题,也需要对整个服务进行扩展。如果使用较小的多个服务,则可以只对需要扩展的服务进行扩展,这样就可以把那些不需要扩展的服务运行在更小的、性能稍差的硬件上。



## 简化部署

在有几百万行代码的单块应用程序中，即使只修改了一行代码,也需要重新部署整个应用程序才能够发布该变更。这种部署的影响很大、风险很高,因此相关千系人不敢轻易做部署。于是在实际操作中,部署的频率就会变得很低。这意味着在两次发布之间我们对软件做了很多功能增强,但直到最后一刻才把这些大量的变更一次性发布到生产环境中。这时,另外一个问题就显现出来了:两次发布之间的差异越大,出错的可能性就更大在微服务架构中,各个服务的部署是独立的这样就可以更快地对特定部分的代码进行部署。如果真的出了问题,也只会影响一个服务,并且容易快速回滚,这也意味着客户可以更快地使用我们开发的新功能。 Amazon和 Netflix等组织采用这种架构主要就是基于上述考虑。这种架构很好地清除了软件发布过程中的种种障碍。微服务部署领域的技术在过去几年时间里发生了巨大的变化,第6章会对该话题做更深入的讨论。

## 与组织结构相匹配

我们经历过太多由于团队和代码库过大引起问题的情况。当团队是分布式的时候,问题会更明显。我们也知道在小型代码库上工作的小团队更加高效。

## 可组合性/组件化

分布式系统和面向服务架构声称的主要好处是易于重用已有功能。而在微服务架构中,根据不同的目的,人们可以通过不同的方式使用同一个功能,在考虑客户如何使用该软件时这一点尤其重要。单纯考虑桌面网站或者移动应用程序的时代已经过去了。现在我们需要考虑的应用程序种类包括Web、原生应用、移动端Web、平板应用及可穿戴设备等,针对每一种都应该考虑如何对已有功能进行组合来实现这些应用。现在很多组织都在做整体考虑,拓展他们与客户交互的渠道,同时也需要相应地调整架构来辅助这种变化的发生。在微服务架构中,系统会开放很多接缝供外部使用。当情况发生改变时,可以使用不同的方式构建应用,而整体化应用程序只能提供一个非常粗粒度的接缝供外部使用。如果想要得到更有用的细化信息,你需要使用榔头撬开它!第5章会讨论如何将已有的单块应用程序分解成为多个微服务,并且达到可重用、可组合的目的。

## 对可替代性的优化

如果你在一个大中型组织工作,很可能接触过一些庞大而丑陋的遗留系统。这些系统无人敢碰,却对公司业务的运营至关重要。更糟糕的是,这些程序是使用某种奇怪的Fortran变体编写的,并且只能运行在25年前就应该被淘汰的硬件上。为什么这些系统直到现在还没有被取代?其实你很清楚答案工作量很大,而且风险很高。当使用多个小规模服务时,重新实现某一个服务或者是直接删除该服务都是相对可操作的。想想看,在单块系统中你是否会在一天内删掉上百行代码,并且确信不会引发问题?微服务中的多个服务大小相似,所以重写或移除一个或者多个服务的阻碍也很小。使用微服务架构的团队可以在需要时轻易地重写服务,或者删除不再使用的服务。当个代码库只有几百行时,人们也不会对它有太多感情上的依赖,所以很容易替换它。

## 其他

* 1）每个微服务相对较小，开发者易于理解， IDE处理效率快，利于提高劳动生产率，Web容器压力小，容器启动速度快，易于提供劳动生产率和生产环境部署速度。
* 2）每个微服务都可以独立部署，简化了部署新服务版本的流程。
* 3）于规模化开发，多个开发团队可以并行开发，每个团队负责一项服务。这意味着，对于开发这而言，仅需要了解自己负责的一部分业务逻辑即可，然而这对于开发者并不一定是好事情。

4）改善故障隔离。一个服务宕机不会影响其他的服务。

1. 接口设计原则

**1.单一职责原则**

Single Responsibility Principle, 简称SRP。

定义：There should never be more than one reason for a class to change.

应该有且仅有一个原因引起类的变更。

职责的划分？单一的定义和级别？

应该根据实际业务情况而定。关注变化点。

实际使用时，类很难做到职责单一，但是接口的职责应该尽量单一。

**2.里氏替换原则**

Liskov Substitution Principle, 简称LSP。

定义：Functions that use pointers or references to base classes must be able to use objects of derived classes without knowing it.

(所有引用基类的地方必须能透明地使用其子类的对象)

里氏替换原则为良好的继承定义了一个规范：

1.子类必须完全实现父类的方法

2.子类可以有自己的个性（属性和方法）。

3.覆盖或实现父类的方法时输入参数可以被放大。

4.覆写或实现父类的方法时输出结果可以被缩小。

**3.依赖倒置原则**

1.高层模块不应该依赖低层模块，两者都应该依赖其抽象。

2.抽象不应该依赖细节。

3.细节应该依赖抽象。

精简的定义： 面向接口编程。

Test-Driven Development 测试驱动开发是依赖倒置原则的最好体现。

测试驱动开发要求先写测试类，测试通过才写实现类，这就要求你要先想接口定义。

 依赖的三种写法：

1.构造函数传递依赖对象。

2.Setter方法传递依赖对象。

3.接口声明依赖对象。

最佳实践：  
1.每个类尽量都有接口或抽象类，或者抽象类和接口两者都具备。  
2.变量的表面类型尽量是接口或抽象类。  
3.任何类都不应该从具体类派生。  
4.尽量不要覆写基类的方法。  
5.结合里氏替换原则使用。

**4.接口隔离原则：**  
接口--这里指用interface关键字定义的接口。  
定义：  
1.Clients should not be forced to depend upon interfaces that they don't use.(客户端不应该依赖它不需要的接口)  
2.The dependency of one class to anther one should depend on the smallest possible interface.(类间的依赖关系应该建立在最小的接口上)

概括：建立单一接口，不要建立臃肿庞大的接口。

通俗来讲：接口尽量细化，同时接口中的方法尽量少。

没有统一的标准，应根据业务合理细分，适合业务才是重点。

保证接口的纯结性：  
1.接口要尽量小。  
2.接口要高内聚。  
3.定制服务。  
4.接口的设计是有限度的。

**5.迪米特法则**  
Law of Demeter, LOD。又称最少知识原则（Least Knowledge Principle, LKP）。  
通俗来讲：一个类应该对自己需要耦合或调用的类知道得最少，你（被耦合或调用的类）的内部是如何复杂都和我没有关系，那是你的事情，我就调用你提供的public方法，其他一概不关心。

低耦合要求：  
1.只和朋友交流  
朋友类：出现在成员变量、方法的输入输出参数中的类。方法体内部的类不属于朋友类。  
2.朋友间也是有距离的  
迪米特法则要求类“羞涩”一点，尽量不要对外公布太多的public方法和非静态的public变量，尽量内敛，多使用private、package-private、protected等访问权限。  
3.是自己的就是自己的  
如果一个方法放在本类中，既不增加类间关系，也对本类不产生负面影响，就放置在本类中。  
4.谨慎使用Serializable

**6.开闭原则**  
Software entities like classes, modules and functions should be open for extension but closed for modifications.（一个软件实体如类、模块和函数应该对扩展开放，对修改关闭）

软件实体包括以下几个部分：  
1.项目和软件产品中按照一定的逻辑规则划分的模块。  
2.抽象和类。  
3.方法。

变化的三种类型：  
1.逻辑变化  
2.子模块变化  
3.可见视图变化