复习题

选择题（均是多选）

1、完整实现EJB规范的产品有（ BC ）。

A、Spring B、GlassFish

C、Tomee D、MyBatis

2、Spring中用于处理HTTP方法的资源注解是（ AD ）。

A、@PostMapping B、@ID

C、@RequestBody D、@PatchMapping

3、Java NIO中，下列哪个可供选择器监控的通道IO事件类型：（ ACD ）。

A、SelectionKey.OP\_READ B、SelectionKey.DENY

C、SelectionKey.CONNECT D、SelectionKey.WRITE

4、Java反射机制API的关键类是（ BCD ）。

A、java.lang.Object B、java.lang.reflect.Method

C、java.lang.Class D、java.lang.reflect.Constructor

1. 下列那些字段可以使得HTTP GET请求就具体化为条件获取（ CD ）。

A、If-Recive-Since B、If-Unmodified-Content

C、If-Match D、If-Match-Date

6、Java Servlet中用于声明一个类为Servlet的注解是（ A ）。

A、@WebServlet B、@Controller

C、@Get D、@Post

7、 HTML DOM模型中，获取网页中元素的方法有（ ABCD ）。

A、getElementById()

B、getElementsByName()

C、getElementByclassName()

D、getElementByTagName()

8、下列属于Java通道类型的是（ BD ）。

A、ServerBootstrap

B、NioSocketChannel

C、NioEventLoop

D、AsynchronousSocketChannel

1. 在HTTP响应消息中，下列哪种状态码可以表示请求资源状态未改变（ D ）。

A、302 B、204

C、102 D、304

10、下列说法正确的是（ ACD ）。

A、HTTP/2.0的消息的以二进制数据进行传输的，结构比复杂。

B、SOAP协议的依托协议是HTTP/2。

C、Netty不能采用阻塞方式工作。

D、gRPC协议是明文的，在消息传递过程中，结构比较简单。

11、属于消息中间件产品的是（ AB ）。

A、ActiveMQ B、RocketMQ

C、Dubbo3 D、MyBatis

12、下列那组件实现了TCP/IP协议规范（ AB ）。

A、Socket B、Java Socket

C、WinSock D、Jfinal

13、Java声明自定义注解的关键字是（ A ）。

A、@interface B、class

C、interface D、final

14、下面哪个属于HTTP/1.1协议的标准METHOD（ AD ）。

A、OPTIONS B、SELECT

C、UPDATE D、PUT

15、指明HTTP响应中包含托挂的关键字是（ D ）。

A、CheckNum B、ExtraInfo

C、Content-Trail D、Trailer

16、JavaNIO中，实现服务器的核心组件是（ B ）。

A、ServerSocket B、ServerSocketChanel

C、Socket D、SocketChanel

17、HTTP/1.1协议Request的首行不包括哪类字段（ B ）。

A、Method

B、Status-Code

C、Request-URI

D、HTTP-Version

18、HTTP协议的消息主体内容编码有（ AB ）。

A、gzip B、compress

C、utf-8 D、en

19、实现跨域请求的方式有（ CD ）。

A、隧道协议 B、PPTP

C、JSONP D、Web代理

20、实现Java Servlet规范的产品有（ ABC ）。

A、TOMEE B、GlassFish

C、Apache2 D、nginx

21、属于数据库中间件产品的是（ ACD ）。

A、ActiveMQ B、SQLAlchemy

C、MTS D、MyBatis

22、在三层结构的分布式系统中，包含了系统的核心业务逻辑的是（ D ）。

A、数据层 B、表示层

C、客户层 D、中间层

23、Java接口文件的不应该出现内容是：（ C ）。

A、interface B、public

C、函数的实现 D、String

24、下面哪一个不属于HTTP/1.1协议的标准METHOD（ C ）。

A、OPTIONS B、CONNECT

C、UPDATE D、PUT

25、下面属于RPC技术和框架有（ AC ）。

A、Dubbo/Dubbox B、SOAP

C、Finagle D、MINA

26、Java 8版本的Java SE提供11种内置注解，请问下列哪些不再其中（ D ）。

A、@Runtime B、@FuntionInterface

C、@SuppressWarnings D、@Table

27、HTTP/1.1协议Response的状态行不包括哪类字段（ AC ）。

A、Method

B、Status-Code

C、Request-URI

D、HTTP-Version

28、关于反射机制下列描述不正确的是（ D ）。

A、在运行时判断任意一个对象所属的类

B、在运行时判断任意一个类所具有的成员变量和方法

C、生成动态代理

D、Java反射的主要实现包含在javax.lang.reflect包中

29、下列哪种不是Java I/O的标准实现方式（ C ）。

A、BIO B、AIO

C、Netty D、NIO

30、下列说法正确的是（ C ）。

A、RPC不能采用UDP作为其底层通信协议

B、SOAP协议的底层协议是TCP/IP

C、RPC 是一种技术思想而非一种规范或协议

D、RPC服务器启动时主动向注册中心登记自己的服务种类、服务方法、版本

31、实现Java Servlet规范的产品有（ BC ）。

A、Hibernate B、GlassFish

C、Tomee D、MyBatis

32、Spring中可以处理指定URL中包含的一个或多个参数的注解是（ A ）。

A、@PathVariable B、@ID

C、@RequestBody D、@Data

33、属于Java AIO中核心的类是：（ A ）。

A、CompletionHandler B、ServerSocket

C、HashMap D、Socket

34、下面哪一个不属于标准HTTP/1.1协议头的参数的是（ D ）。

A、Last-Modified B、User-Agent

C、Connection D、Length

35、下面不属于RPC框架的是（ D ）。

A、Dubbo/Dubbox B、gRPC

C、Finagle D、MINA

36、Java 8版本的Java SE提供5种内置注解，请问下列哪些是正确的（ BC ）。

A、@Runtime B、@Funtioninterface

C、@SuppressWarnings D、@Table

38、下列属于Netty通道类型的是（ B ）。

A、ServerBootstrap

B、NioSocketChannel

C、NioEventLoop

D、AsynchronousSocketChannel

39、在HTTP响应消息中，下列哪种状态码可以表示请求已经成功处理（ B ）。

A、302 B、204

C、102 D、201

40、下列说法正确的是（ B ）。

A、HTTP/1.1的消息的以二进制数据进行传输的，结构比复杂。

B、SOAP协议的依托协议是HTTP。

C、Netty不能采用阻塞方式工作。

D、HTTP/2协议是明文的，在消息传递过程中，结构比较简单。

填空题

1、 软件的三大类型：单机类型、BS类型、CS类型

2、 反射（Reflection）是程序的\_\_\_\_自我分析能力\_\_\_\_\_\_，通过反射可以确定类有哪些方法、有哪些构造方法以及有哪些成员变量。Java语言提供了反射机制，通过反射机制能够\_\_\_\_\_\_\_\_动态读取一个类的信息\_\_\_\_\_\_\_；能够在运行时动态加载类，而不是在\_\_\_\_编译期\_\_\_\_\_。反射可以应用于框架开发，它能够从配置文件中读取配置信息动态加载类、\_\_\_\_\_\_\_创建对象\_\_，以及调用方法和成员变量。

3、 HTTP/1.1协议的响应消息由\_\_\_\_响应消息头\_\_\_\_、\_\_响应状态行\_\_\_\_\_\_和\_\_\_实体内容\_\_\_\_\_组成。

4、 实现ORM中间件重点处理\_\_\_\_\_\_\_\_\_、\_\_\_\_\_\_\_\_\_\_、\_\_\_\_\_\_\_\_\_\_自动化。

5、 Java NIO的三个核心组件是：*Channel(通道),Buffer(缓冲区),Selector(选择器)*

6、 在HTTP/1.1协议中，处理多事务的连接的方式有：\_\_\_\_\_\_\_\_\_、\_\_\_\_\_\_\_、\_\_\_\_\_\_\_、\_\_\_\_\_\_。

7、 常见的五种基本I/O模型，分别是：阻塞I/O模型、非阻塞I/O模型、多路复用I/O模型、信号驱动I/O模型、异步I/O模型。

8、Java 8提供的元注解包括：@Documented、@Target、@Retention、\_\_\_\_@Inherited\_\_\_\_、\_\_@Repetable\_\_\_\_和\_\_@Native\_\_\_\_。

9、一个HTTP事务由一条\_\_请求命令\_\_\_\_\_和一条\_\_\_响应结果\_\_\_\_\_\_构成。

10、Reactor模式由Reactor反应器线程、Handlers处理器两大角色组成。

11、 I/O通信模型有：（和7一样）

12、 Java BIO和NIO的本质区别是，Java BIO是面向\_\_\_流\_\_\_，NIO是面向\_\_\_\_块（缓冲区）\_\_\_。

13、 Java NIO的核心组件分别是：Channel(通道)、Buffer(缓冲区)、Selector(选择器)组成。

14、 HTTP响应的状态行由HTTP版本(HTTP-Version)、状态码(Status-Code)以及解释状态码的简单短语(Reason-phrase)三部分构成。

15、 Netty的Handler分为两大类：通道入站处理器Inbound\_\_\_\_\_\_\_\_\_和\_\_\_\_\_\_通道出站处理器Outbound\_\_\_。

16、 在HTTP/1.1协议中，处理多事务的连接的方式有：\_\_\_\_\_\_\_\_\_、\_\_\_\_\_\_\_、\_\_\_\_\_\_\_、\_\_\_\_\_\_。

17、HTTP协议中，处理多事务的连接有\_\_\_\_\_\_\_、\_\_\_\_\_\_\_、\_\_\_\_\_\_\_和\_\_\_\_\_\_\_。

18、Java 8提供的元注解包括：见8

19、浏览器将CORS请求分成两类：简单请求（simple request）和非简单请求（not-so-simple request）。

简答

1. 在Java中，NIO和BIO的区别主要体现在那三个方面？

答： 在Java中，BIO是面向流的，NIO是面向缓冲区（块）的;

BIO的各种流是同步阻塞的，而NIO是同步非阻塞的。

BIO没有选择器的概念，NIO有选择器的概念，NIO会等待数据全部传输过来以后再开辟一个线程去处理，而BIO会一有请求就开辟一个线程，如果数据没有完成传输，就让它等待

1. RESTful架构风格背后的主要思想是什么？

答：：统一接口：组件之间有统一的接口。

RESTful web服务是基于客户机/服务器的，不存储状态。从客户机到服务器的请求的结果可以缓存在客户机中。它具有基于超媒体的统一接口，具有自描述消息。而且客户机和服务器并不知道它们之间的中间连接。

1. AJAX中包括哪些技术？

答; Ajax即Asynchronous Javascript And XML（异步JavaScript和XML），是一种Web数据交互方式，一种支持异步请求的技术。

基于Web标准(XHTML + CSS)的展示；

使用DOM进行动态显示和交互；

使用XMLHttpRequest进行数据交换和相关操作；

用JS可以控制XMLHttpRequest对象向服务器提出请求并处理响应，而不影响用户对页面的正常访问。

1. 请简述HTTP/1.1协议300型响应码中几个重要的响应码，含义是什么？

答：300 Multiple Choices 客户端请求了实际指向多个资源的URL。

301 Moved Permanently 请求的 URL 已移走。

302 Found 请求的URL临时移走

303 See Other 客户端应该使用指定URL

307 Temporary Redirect 客户端应该临时定位到指定URL

于是，看起来并不复杂的重定向，稍微对比就会感到很混淆的。特别是301/302 ，303/307 一组，好像根本就是重复的。

5、 “序列化”和“反序列化”会涉及POJO的编码和格式化（Encoding & Format），目前我们可选择的编码方式有几种？评价一个序列化框架的优缺点应该从那些方面着手？

6、简述HTTP协议请求的八个方法的名称及其功能。

GET 表示我们要请求一个由URI指定的在服务器上的资源。

PUT方法 表示如果指定URL资源不存在就创建它，否则就修改它。

POST方法 表示要创建一个新的子资源，或者更新一个存在的资源。

DELETE表示我们要删除一个由URI指定的资源。

HEAD 和GET一样，但是仅仅返回指定资源响应的头部分，而不必返回响应主体

OPTIONS 查询目标资源支持method的清单。

TRACE 查询到目标资源经过的中间节点。用于测试。

CONNECT 建立一个到URI指定的服务器的隧道。

7、简述java.lang.reflect包中提供反射处理能力的几个类并说明其作用。

java.lang.reflect包提供了反射中用到类，主要的类说明如下：

Constructor类：提供类的构造方法信息。

Field类：提供类或接口中成员变量信息。

Method类：提供类或接口成员方法信息。

Array类：提供了动态创建和访问Java数组的方法。

Modifier类：提供类和成员访问修饰符信息。

8、简述目前Java中常见的几种实现RESTful技术方案。

使用Spring的RestTemplate

通过Servlet实现

JAX-RS

JAX-RS即Java API for RESTful Web Services

9、请简述java.lang.Class类和java.lang.Object的作用和区别。

Class类也是类的一种，只是名字和class关键字高度相似，用于java反射机制。

Class类的对象不能像普通类一样，以 new 的方式创建，它的对象只能由JVM创建，因为这个类没有public构造函数

Class类的作用是运行时提供或获得某个对象的类型信息，这些信息也可用于反射。

Object类是一切java类的父类，对于普通的java类，即便不声明，也是默认继承了Object类。

Object类位于java.lang包中，java.lang包包含着Java最基础和核心的类，在编译时会自动导入

Object类定义了线程同步与交互的一系列方法

10、在Java中，NIO和BIO的区别主要体现在那三个方面？

见1

11、 简述Java NIO中使用Buffer类的基本步骤？

12、 AJAX中包括哪些技术？

Ajax即Asynchronous Javascript And XML（异步JavaScript和XML），是一种Web数据交互方式，一种支持异步请求的技术。

基于Web标准(XHTML + CSS)的展示

使用DOM进行动态显示和交互

使用XMLHttpRequest进行数据交换和相关操作

用JS可以控制XMLHttpRequest对象向服务器提出请求并处理响应，而不影响用户对页面的正常访问。

13、 请简述HTTP/1.1协议200型响应码中几个重要的响应码，含义是什么？

200 OK：是最常用的状态。它指明请求已经成功完成。

201 Created ：会比200 OK 有更加具体的语义。201指明请求成功且创建了一个资源，因此201常常配合PUT方法使用，因为PUT方法的语义上就是创建一个资源。

202 Accepted ：会比200 OK 有更加具体的语义。202表明请求成功被接受，但不一定已经完成资源创建或者修改，而只是被接受，可能还有服务器的后续的处理。

204 No Content：表明请求处理成功，但是作为服务器并不想要提供消息在消息主体内，或者并没有什么消息主体需要提供。

205 Reset Content：此状态码告诉客户端请求已经成功执行。不同于204，它的意图是要告诉客户应该清除Form的内容或者刷新用户界面。

206 Partical Content：它的存在目的是为了支持大文件的分段下载。当客户端发起资源范围请求，服务器就可以返回206型响应，告知客户端操作成功并且返回部分内容。

14、 简述Reactor模式的优点和缺点

优点：

（1）工作线程一般是提前创建好的，可以循环复用。相比于每次有客户请求过来就新建线程去处理，这种方式大大减少了线程创建和销毁的开销。

（2）可以通过增加工作线程的个数来充分利用CPU资源。

（3）Reactor实现相对简单，对于耗时短的处理场景处理高效。

（4）事务分离：将与应用无关的多路分解和分配机制和与应用相关的回调函数分离开来

操作系统可以在多个事件源上等待，并且避免了多线程编程相关的性能开销和编程复杂性

缺点：

（1）相比传统的简单模型，Reactor增加了一定的复杂性，并且不易于调试。

（2）Reactor模式需要用到系统提供的IO多路复用函数，如select、poll、epoll，如果要自己实现可能不会有那么高效。

（3）Reactor处理耗时长的操作会造成事件分发的阻塞，影响到后续事件的处理；

15、简述HTTP协议200型响应码中常用的响应码及其作用。

见13

16、简述Java反射机制中获取Java类构造方法的函数，并说明其作用。

17、简述目前Java中常见的几种实现RESTful技术方案。

使用Spring的RestTemplate

通过Servlet实现

JAX-RS

JAX-RS即Java API for RESTful Web Services

18、列出Java NIO通道类中我们常用的文件处理和网络通信常用的通道类，并说明起作用。

19、简述哪些HTTP请求会产使用到CORS机制。

**程序综合题**

下面代码是数据库连接缓冲池的实现，其中空缺两个函数没有实现，请根据要求补齐如下两个函数代码，注意合理利用已定义变量：

1、 实现数据库连接缓冲池public void initPool函数代码。

2、 实现数据库连接缓冲池public synchronized Connection getConnection() 的函数代码。

public class DBPool {

/\*\*\* 连接池对象\*/

private List<Connection> idealpool;//空闲缓冲池

private List<Connection> inServingPool;//在服务缓冲池

/\*\*最大最小连接数\*/

private static final int POOL\_MAX\_SIZE = DBManager.getConfiguration().getPoolMaxSize();

private static final int POOL\_MIN\_SIZE = DBManager.getConfiguration().getPoolMinSize();

**public void initPool() {**

if (idealpool == null) {

idealpool = new ArrayList<Connection>();

}

inServingPool = new ArrayList<Connection>();

while (idealpool.size() < DBPool.POOL\_MIN\_SIZE) {

idealpool.add(DBManager.createConnection());

System.out.println("初始化池，池中连接数：" + idealpool.size());

}

}

**public synchronized Connection getConnection() {**

**int last\_index = idealpool.size() - 1;**

**Connection conn = null;**

**if (last\_index >= 0) {**

**conn = idealpool.get(last\_index);**

**idealpool.remove(last\_index);**

**inServingPool.add(conn);**

**}else if (inServingPool.size()<POOL\_MAX\_SIZE) {**

**conn=DBManager.createConnection();**

**inServingPool.add(conn);**

**}**

**return conn;**

**}**

/\*\*

\* 将连接放回池中

\* @param conn

\*/

public synchronized void close(Connection conn) {

if (idealpool.size()+inServingPool.size() >= POOL\_MAX\_SIZE) {

try {

if (conn != null) {

conn.close();

}

} catch (SQLException e) {

e.printStackTrace();

}

} else {

idealpool.add(conn);

inServingPool.remove(conn);

}

}

public DBPool() {

this.initPool();

}

}

1、补齐HTTP Response类程序中的缺失的代码**。**

public class Response {

private StringBuilder headInfo; // 响应行和头信息

private StringBuilder content; // 正文

private BufferedWriter bw;

public int code = \_\_200\_\_;

private int len;

private final String space = " ";

private final String CRLF = "\n";

public Response(OutputStream outputStream) {

headInfo = new StringBuilder();

content = new StringBuilder();

bw = new BufferedWriter(new OutputStreamWriter(outputStream));

len = 0;

}

public void print(String html) throws IOException {

content.append(html);

this.len = html.getBytes().length;

\_\_setHead()\_\_;

bw.append(headInfo.toString());

bw.append(content.toString());

bw.flush();

}

// 构建响应行和响应头

private void setHead() {

if (code != 200)

this.len = 0;

headInfo.append("HTTP/1.1").append(space).append(code).append(space);

switch (code) {

case 200:

headInfo.append("\_ok\_");

break;

case 404:

headInfo.append("\_\_NOT FOUND\_");

break;

case 505:

headInfo.append("\_\_Server ERROR\_\_");

break;

}

headInfo.append(\_\_CRLF\_\_);

headInfo.append("\_\_Date\_").append(LocalDateTime.now()).append(CRLF);

headInfo.append("\_\_Content-Type\_\_:text/html;charset=UTF-8").append(CRLF);

headInfo.append("\_Content-Length\_").append(space).append(this.len).append(CRLF).append(CRLF);

}

//

public void setCode(int code) {

\_\_this.code\_\_ = code;

}

}

下面代码是HTTP Request解析类，请根据协议要求，补齐空格中必须填入的代码，注意合理利用已定义变量：

public class Request {

private String method;

private String url;

private Map<String, List<String>> parameterMap;

private InputStream is;

private final String CRLF = \_\_\_“\n”\_\_\_\_\_;

public Request(InputStream is) {

try {

method = "";

url = "";

parameterMap = new HashMap<>();

this.is = is;

byte[] arr = new byte[20000];

int len = \_\_\_\_\_this.is.read(arr)\_\_\_\_\_;

if (len > 0) {

String requestInfo = \_\_\_\_\_new String(arr, 0, len) \_\_ ;

paraseRequest(requestInfo);

}

} catch (IOException e) {

e.printStackTrace();

}

}

private void paraseRequest(String requestInfo) throws UnsupportedEncodingException {

String param = "";

String firstline = requestInfo.substring(0, requestInfo.indexOf(CRLF));

this.method = firstline.substring(0,firstline.indexOf("/")).trim();

String url = firstline.substring(firstline.indexOf("/"),

firstline.indexOf("HTTP/")).trim();

if (method.equalsIgnoreCase("post")) {

this.url = url;

param = requestInfo.substring(requestInfo.lastIndexOf(CRLF)).trim();

} else if (method.equalsIgnoreCase("get")) {

if (url.contains("?")) {

String[] arr = url.split("\\?");

this.url = \_\_\_arr[0]\_\_\_\_\_;

param = \_\_\_\_arr[1]\_\_\_\_;

} else {

this.url = url;

}

}

if (param != "") {

StringTokenizer token = new StringTokenizer(param,\_"&"\_\_);

while (token.hasMoreElements()) {

String key\_value = (String) token.nextElement();

String[] ks = key\_value.split(\_\_\_\_“=”\_\_\_\_);

if (ks.length == 1) {

ks = Arrays.copyOf(ks, 2);

ks[1] = \_\_\_\_null\_\_\_\_;

}

if (!parameterMap.containsKey(ks[0].trim())) {

parameterMap.put(ks[0].trim(), new ArrayList<String>());

}

parameterMap.get(ks[0].trim()).add(URLDecoder.decode(ks[1].trim(), "UTF-8"));

}

}

}

public String getParameter(String name) {

String[] values = getParameterValues(name);

if (values == null)

return null;

else

return\_\_\_\_values[0]\_\_\_\_\_;

}

public String[] getParameterValues(String name) {

if (parameterMap.get(name) == null) {

return null;

} else {

List<String> values = parameterMap.get(name);

return values.toArray(new String[values.size()]);

}

}

public String getUrl() {

return \_\_\_\_this.url\_\_\_;

}

补齐通用服务线程池ServicePool程序中的缺失的代码**。**

**public** **class** ServicePool<Job **extends** \_\_\_\_23\_\_\_\_> {

**private** **final** LinkedList<Job> jobsList = **new** LinkedList<>();

**private** **final** List<MyWorker> workerList = Collections.*synchronizedList*(**new** ArrayList<MyWorker>());

**private** **static** **final** **int** ***DEFAULT\_WORKER\_NUMBERS*** = 5;

**private** AtomicLong threadNum = **\_\_\_\_\_24\_\_\_\_\_\_\_**;

**public** ServicePool() {

initWorkerThreadByNum(***DEFAULT\_WORKER\_NUMBERS***);

}

**public** ServicePool(**int** workerNum) {

initWorkerThreadByNum(workerNum);

}

**public** **void** initWorkerThreadByNum(**int** workerNum) {

**for** (**int** i = 0; i < workerNum; i++) {

MyWorker worker = **new** MyWorker();

\_\_\_\_\_\_\_25\_\_\_\_\_\_\_\_;

**new** Thread(worker, "ThreadPool-Worker-" + threadNum.incrementAndGet()).start();

}

}

**public** **void** addJob(Job job) {

**if** (job != **null**) {

**synchronized** (jobsList) {

jobsList.addLast(job);

System.***out***.println("剩余待处理请求个数：" + ServicePool.**this**.getJobsize());

\_\_\_\_\_26\_\_\_\_\_\_;

}

}

}

**public** **void** shutdown() {

**for** (MyWorker e : workerList) {

e.shutdown();

}

}

**public** **int** getJobsize() {

**return** \_\_\_\_\_27\_\_\_\_\_;

}

**private** **class** MyWorker **implements** Runnable {

**private** **\_\_\_\_\_28\_\_\_\_\_\_** **boolean** isRunning = **true**;

@Override

**public** **void** run() {

**while** (\_\_\_\_29\_\_\_\_\_) {

Job job = **null**;

**synchronized** (jobsList) {

**while** (jobsList.isEmpty()) {

**try** {

\_\_\_\_\_30\_\_\_\_\_;

} **catch** (InterruptedException e) {

**return**;

}

}

job = \_\_\_\_\_31\_\_\_\_\_;

}

**if** (job != **null**) {

System.***out***.println("正在处理请求");

\_\_\_\_\_32\_\_\_\_\_;

System.***out***.println("处理完成，剩余待处理请求个数：" + ServicePool.**this**.getJobsize());

}

}

}

**public** **void** shutdown() {

isRunning = **false**;

}

}

}

**分析题(共10分)**

根据下图1，详述说明RPC框架中每个部分的功能。

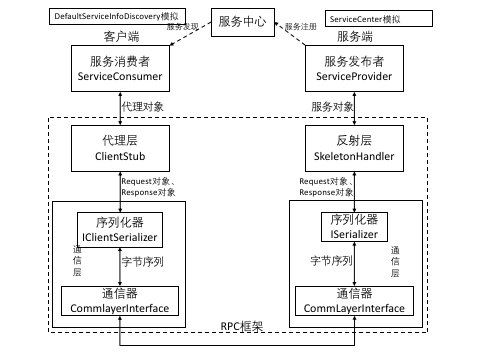


图1 RPC框架结构图

**(1)、服务发布者：运行在服务器端，提供服务接口定义与服务实现类。**

**(2)、服务中心：运行在服务器端，负责将本地服务发布成远程服务，管理远程服务，提供给服务消费者使用。**

**(3)、服务消费者：运行在客户端，通过远程代理对象调用远程服务。**

**服务提供者启动后主动向服务（注册）中心注册机器ip、端口以及提供的服务列表； 服务消费者启动时向服务（注册）中心获取服务提供方地址列表。**