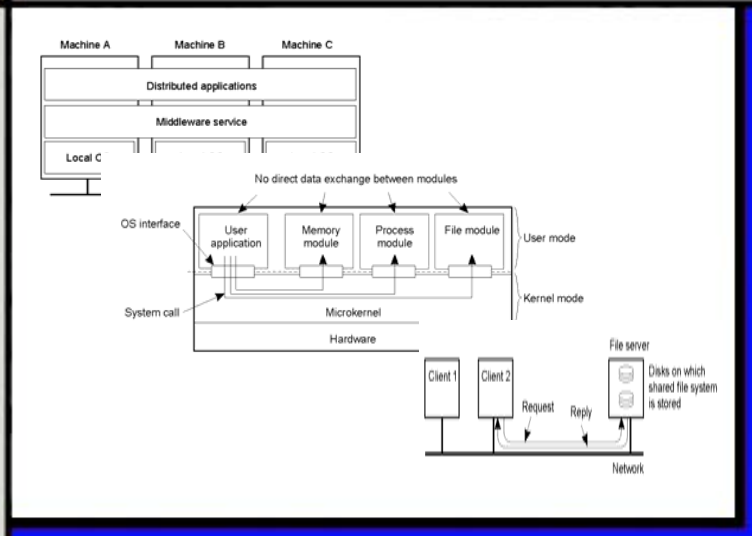
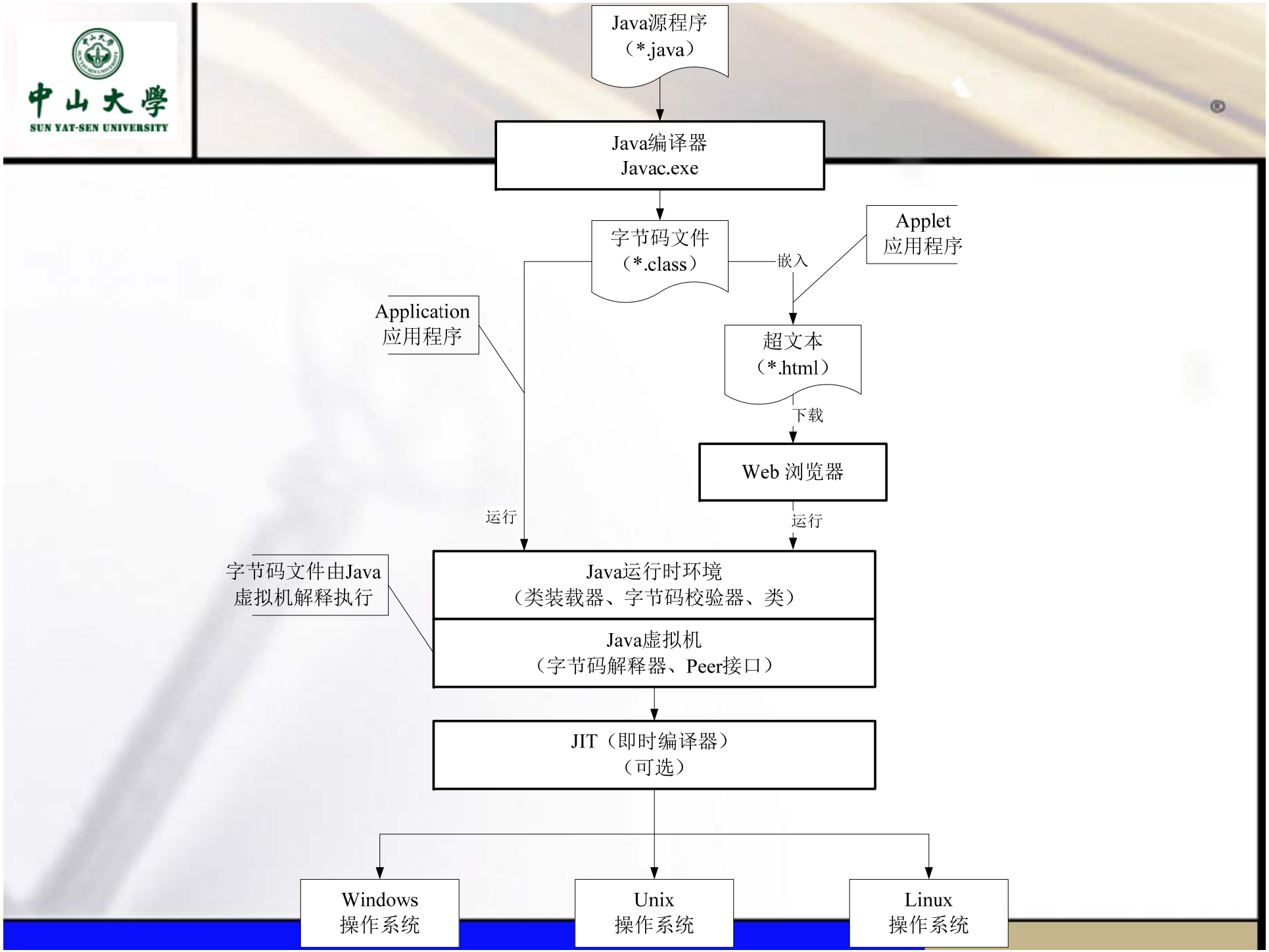
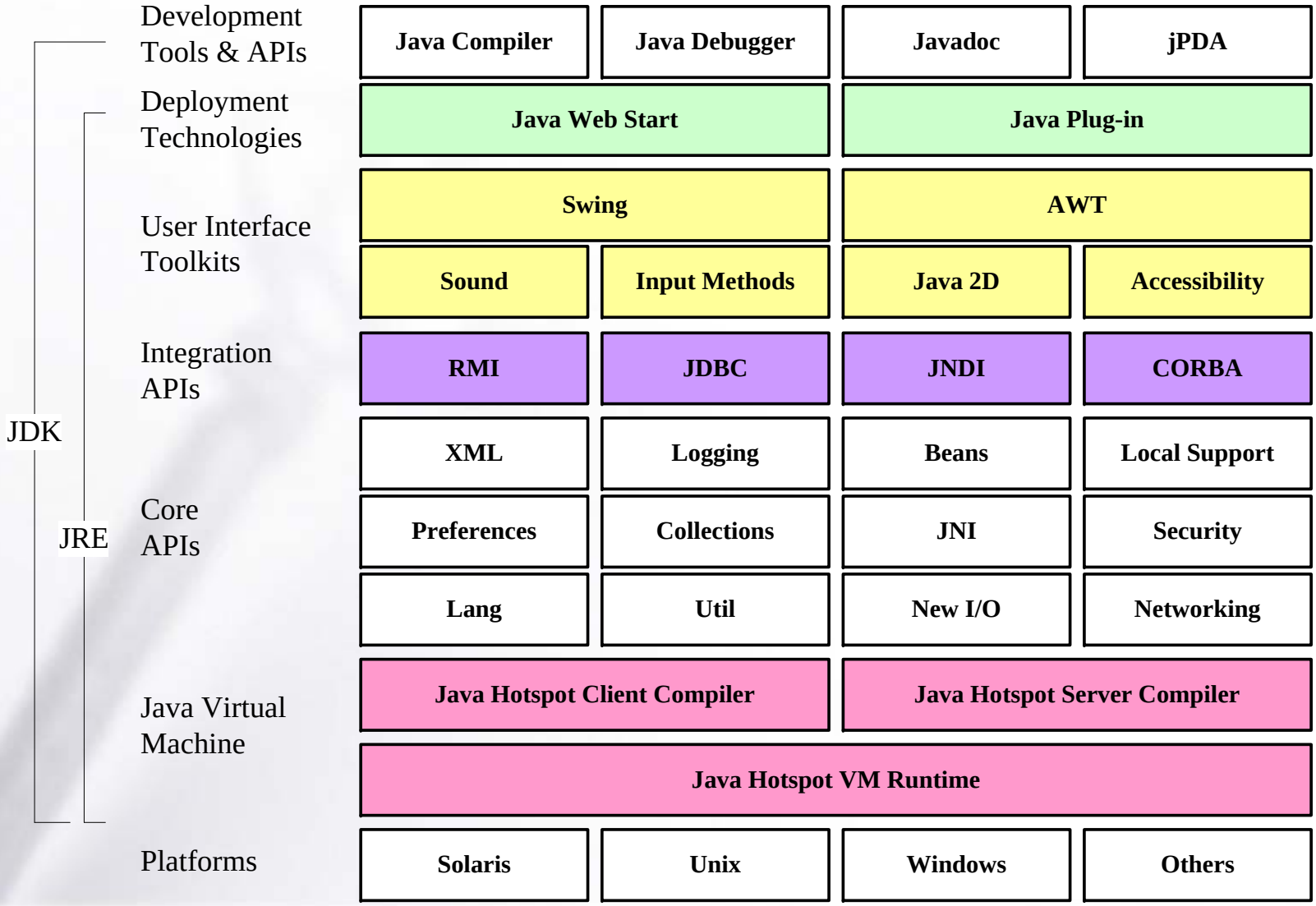
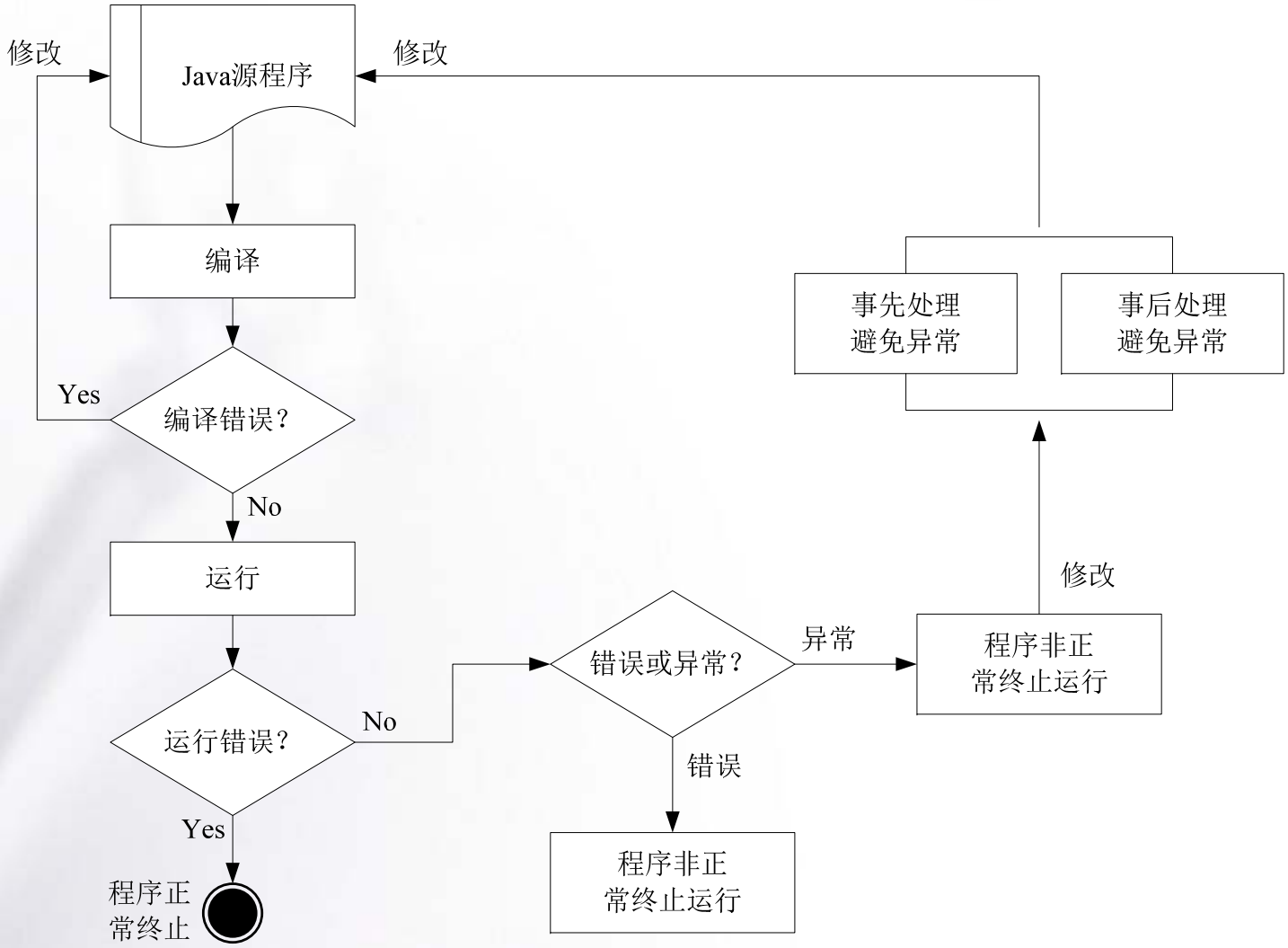
中山大學
SUN YAT-SEN UNIVERSITY

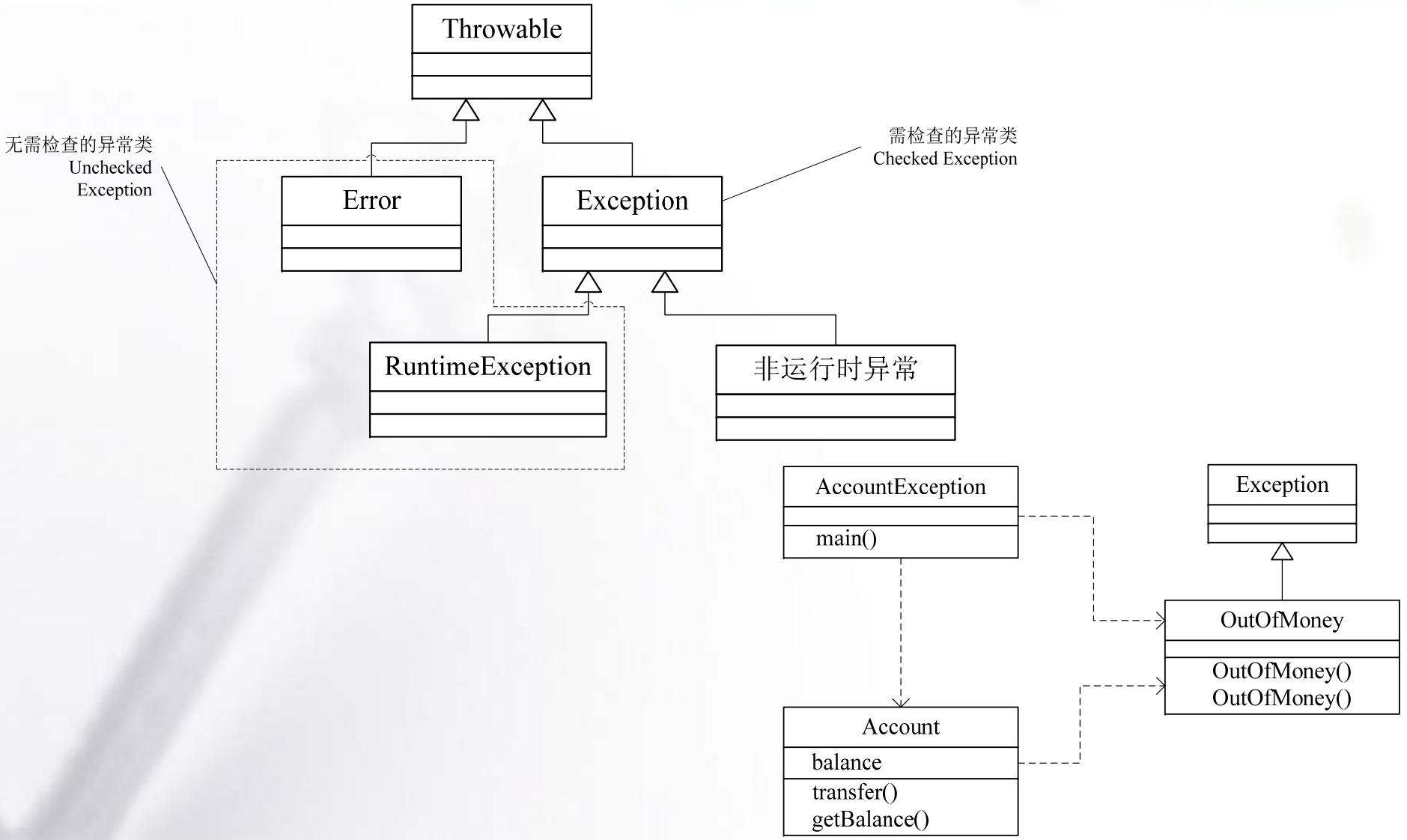


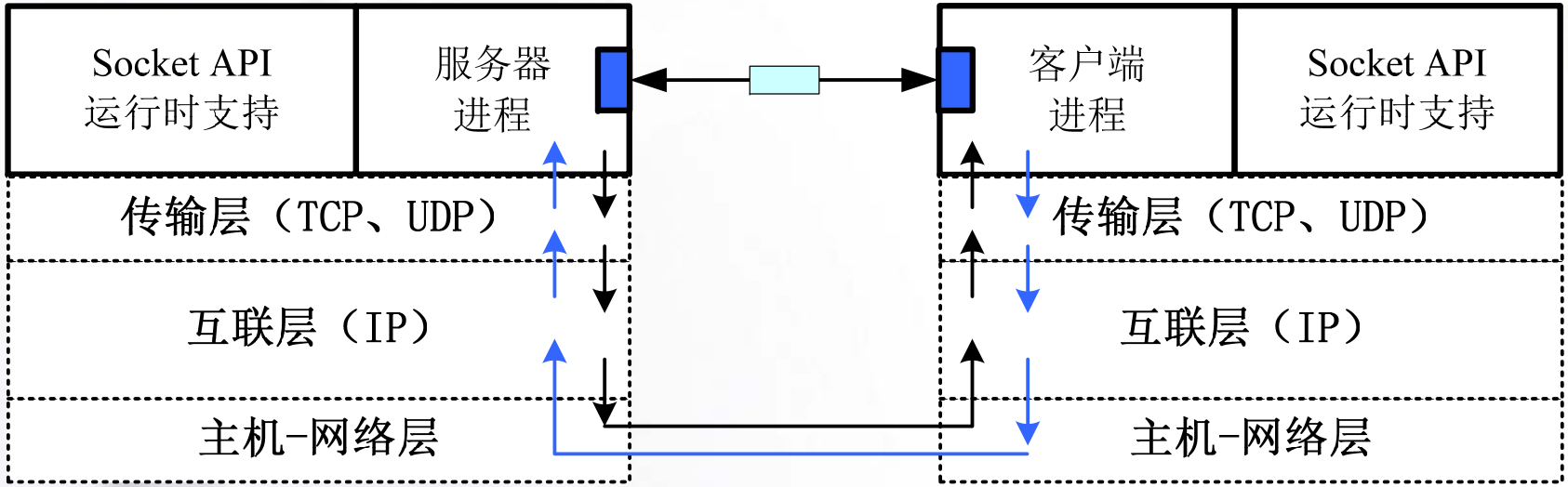
Distributed Computing

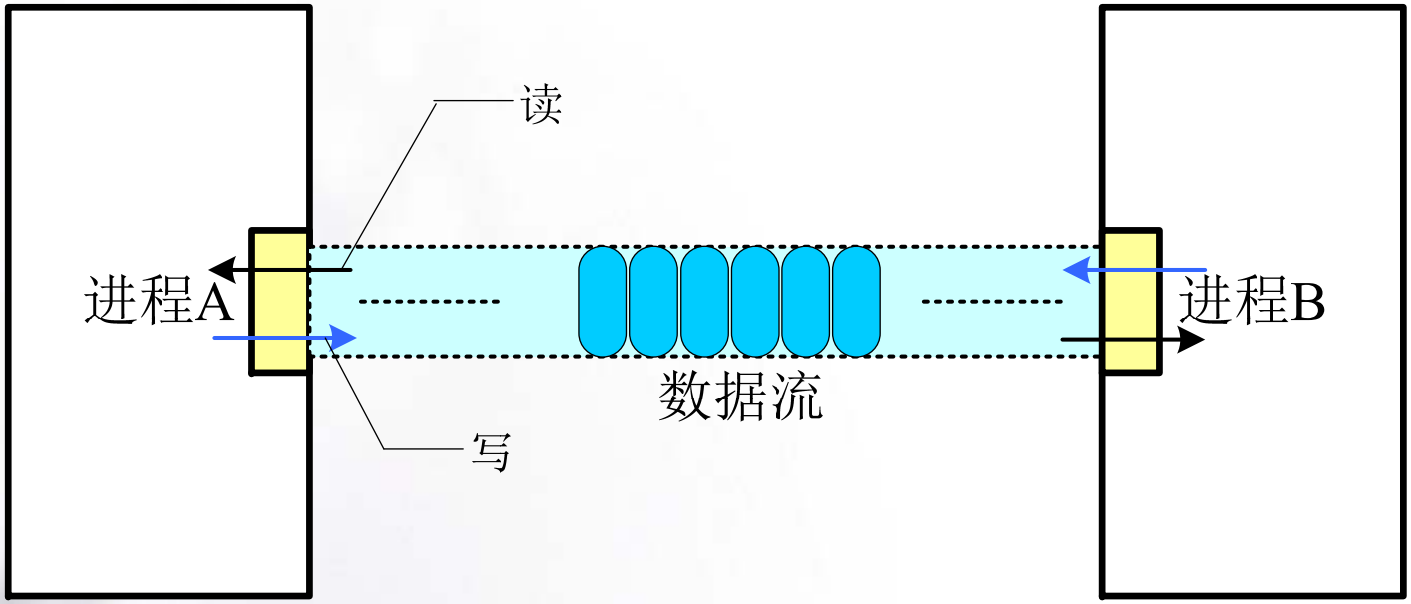

Java源程序
（*.java）
Java编译器
Javac.exe
字节码文件
（*.class）
嵌入
Applet
应用程序
Application
应用程序
超文本
（*.html）
下载
Web 浏览器
运行
运行
字节码文件由Java
虚拟机解释执行
Java运行时环境
（类装载器、字节码校验器、类）
Java虚拟机
（字节码解释器、Peer接口）
JIT（即时编译器）
（可选）
Windows
操作系统
Unix
操作系统
Linux
操作系统

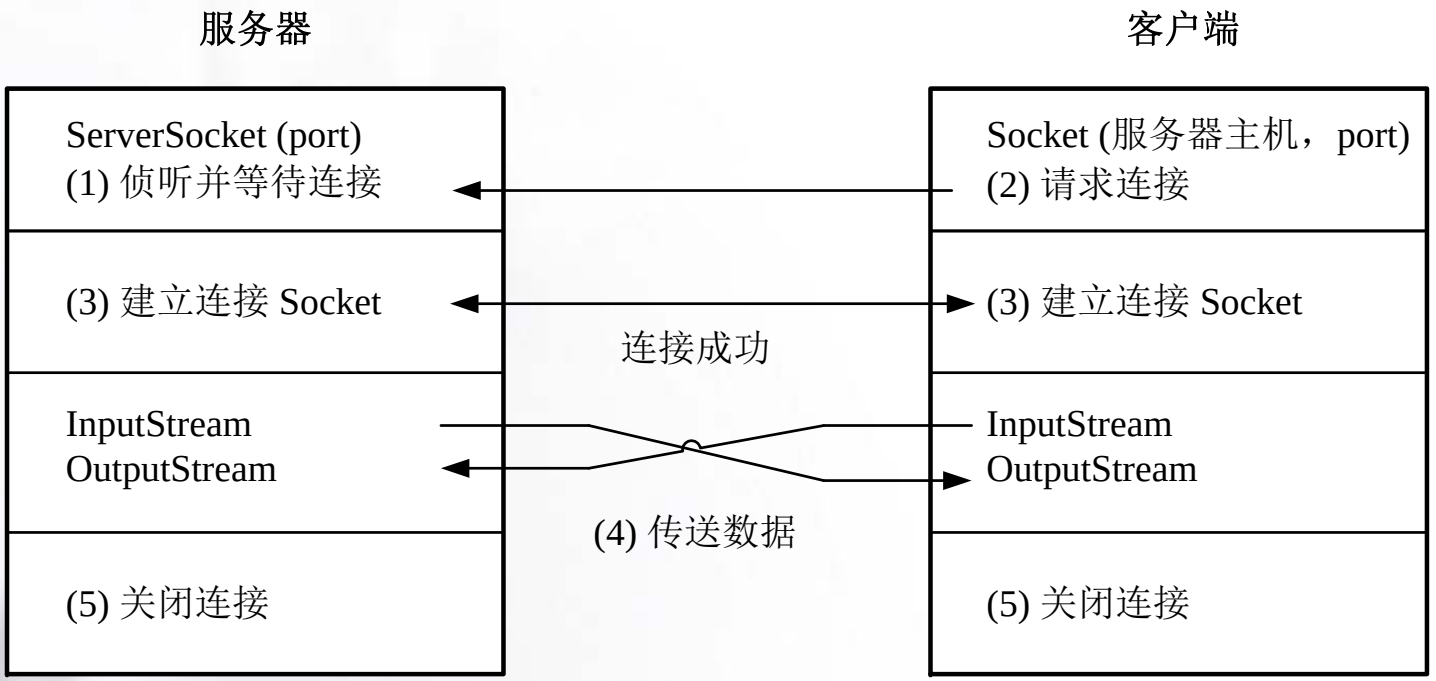

JDK
JRE
Development Tools & APIs
Java Compiler
Java Debugger
Javadoc
jPDA
Deployment Technologies
Java Web Start
Java Plug-in
User Interface Toolkits
Swing
AWT
Sound
Input Methods
Java 2D
Accessibility
Integration APIs
RMI
JDBC
JNDI
CORBA
Core APIs
XML
Logging
Beans
Local Support
Preferences
Collections
JNI
Security
Lang
Util
New I/O
Networking
Java Virtual Machine
Java Hotspot Client Compiler
Java Hotspot Server Compiler
Java Hotspot VM Runtime
Platforms
Solaris
Unix
Windows
Others


修改
Java源程序
修改
编译
事先处理
避免异常
事后处理
避免异常
Yes
编译错误?
No
运行
修改
异常
错误或异常?
程序非正
常终止运行
No
运行错误?
错误
Yes
程序正
常终止
程序非正
常终止运行


Throwable
无需检查的异常类
Unchecked
Exception
需检查的异常类
Checked Exception
Error
Exception
RuntimeException
非运行时异常
AccountException
main()
Exception
OutOfMoney
OutOfMoney()
OutOfMoney()
Account
balance
transfer()
getBalance()


Socket API
运行时支持
服务器
进程
客户端
进程
Socket API
运行时支持
传输层（TCP、UDP）
传输层（TCP、UDP）
互联层（IP）
互联层（IP）
主机-网络层
主机-网络层


读
进程A
进程B
数据流
写


服务器
客户端
ServerSocket (port)
(1) 侦听并等待连接
Socket (服务器主机，port)
(2) 请求连接
(3) 建立连接 Socket
(3) 建立连接 Socket
连接成功
InputStream
OutputStream
InputStream
OutputStream
(4) 传送数据
(5) 关闭连接
(5) 关闭连接

程序 2-4 RunnableThread.java例程

```
// 通过实现Runnable接口定义新线程
class Counter implements Runnable {
        public void run() {
                for (int i = 0; i < 100; i++) System.out.println("计数器 =  " + i);
        }
}
public class RunnableThread {
        public static void main(String[] args) {
                Counter counter = new Counter();
                Thread thread = new Thread(counter);
                thread.start();
                System.out.println("主程序结束");
        }
}
```

程序 2-5 SubclassThread.java例程

```
// 通过继承Thread类定义新线程
public class SubclassThread extends Thread {
        public void run() {
                while (true) {
                // 执行线程自身的任务
                        try {
                        sleep(5 * 1000);
                        break;
                        } catch (InterruptedException exc) {
                        // 睡眠被中断
                        }
                }
        }

        public static void main(String[] args) {
                Thread thread = new SubclassThread();
                thread.start();
                System.out.println("主程序结束");
        }
}
```

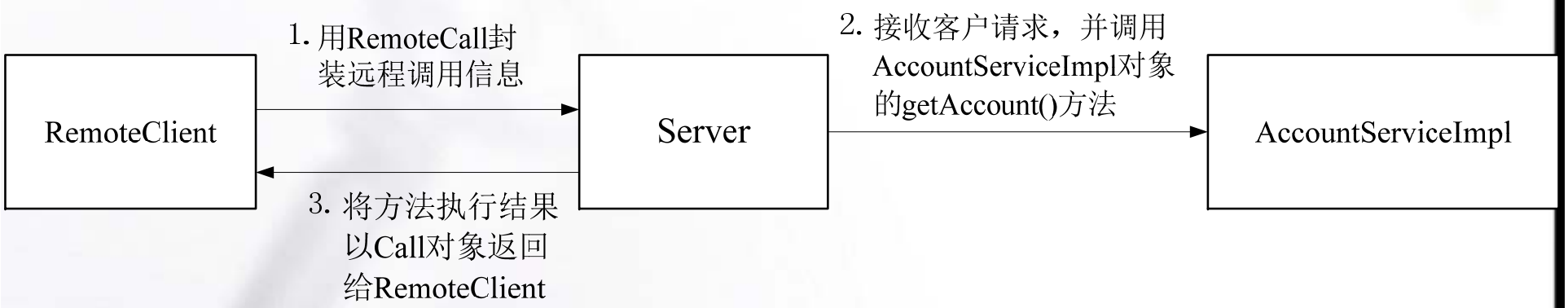
程序 2-6 AccountSerializable.java例程

```
//测试对象的序列化和反序列化
import java.io.*;
public class AccountSerializable{
        public static void main(String[] args) throws Exception {
                //创建本地文件输入流
                File f = new File("objectFile.obj");
                ObjectOutputStream out = new ObjectOutputStream(new FileOutputStream(f));

                //序列化对象
                Account Account1 = new Account("Zhang3", 1000);
                Account Account2 = new Account("Li4", 2000);
                out.writeObject(Account1);
                out.writeObject(Account2);
                out.close();
                //反序列化对象
                ObjectInputStream in = new ObjectInputStream(new FileInputStream(f));
                Account obj1 = (Account) in.readObject();
                System.out.println("Account1=" + obj1);
                Account obj2 = (Account) in.readObject();
                System.out.println("Account2=" + obj2);
                in.close();
        }
}
```

程序 2-6 AccountSerializable.java例程

```
//Account类实现java.io.Serializable接口
class Account implements Serializable {
        private String name;
        private double balance;
        public Account(String name, double balance) {
                this.name = name;
                this.balance = balance;
        }
        public String toString() {
                return "name=" + name + ", balance=" + balance;
        }
}
```


RemoteClient
1. 用RemoteCall封装远程调用信息
Server
2. 接收客户请求，并调用AccountServiceImpl对象的getAccount()方法
AccountServiceImpl
3. 将方法执行结果以Call对象返回给RemoteClient

程序 2-8 AccountService.java例程

```
//远程对象接口
public interface AccountService {
        public String getAccount(String Name);
}
```

程序 2-9 AccountServiceImpl.java例程

```
//远程对象接口的实现类
public class AccountServiceImpl implements AccountService{
        public String getAccount(String Name){
                return "Account id: "+ Name;
        }
}
```

程序 2-10 RemoteCall.java例程

```
//远程调用对象
import java.io.*;
public class RemoteCall implements Serializable{
        private String className; //表示类名或接口名
        private String methodName; //表示方法名
        private Class[] paramTypes; //表示方法参数类型
        private Object[] params; //表示方法参数值
        //表示方法的执行结果
        //如果方法正常执行，则result为方法返回值，如果方法抛出异常，那么result为该异常。
        private Object result;
        public RemoteCall(){}
        public RemoteCall(String className,String methodName,Class[] paramTypes, Object[] params){
                this.className=className;
                this.methodName=methodName;
                this.paramTypes=paramTypes;
                this.params=params;
        }
        public String getClassName(){return className;}
        public void setClassName(String className){this.className=className;}
```