面向对象特性:

继承、封装、多态、抽象

修饰符	当前类	同 包	子 类	其他包
public	√	√	√	√
protected	√	√	√	×
default	√	√	×	×
private	√	×	×	×

对象的克隆:

```
Person p1 = new Person(12, "Tom"); //对象需要实现Cloneable接口并重写clone()方法
Person p2 = (Person) p1.clone();  //clone()属于浅拷贝,要想深拷贝对象，需要在clone()方法里也要拷贝对象中的成员变量
```

&与&&的区别

&叫逻辑与、按位与，&&叫短路与；username != null && !username.equals("") 左边为false时就被短路，不再执行右边，如果用&，则两边都要执行。逻辑或运算符（|）和短路或运算符（||）的差别也是如此。

构造方法不能被重写，声明为final的方法不能被重写，声明为static的方法不能被重写，但能被再次声明。

抽象类和接口的异同

有抽象方法的类必须声明为抽象类，抽象类未必有抽象方法

抽象类中可以包含静态方法，接口中不能包含静态方法

一个类只能继承一个抽象方法，但可以实现多个接口

抽象类中可以定义构造器，接口不能

抽象类个的成员变量可以使private、default、protected、public，接口中的成员变量只能是public static final的全局变量

相同：不能被实例化

Java的基本数据类型

四类	八种	字节数	数据表示范围
整型	byte	1	-128～127
	short	2	-32768～32767
	int	4	-2147483648～2147483647
	long	8	-2^{63}～$2^{63}-1$
浮点型	float	4	-3.403E38～3.403E38
	double	8	-1.798E308～1.798E308
字符型	char	2	表示一个字符，如('a'，'A'，'0'，'家')

short s1 = 1; s1 = s1 + 1;是错误的，因为1是int型。

short s1 = 1; s1 += 1; 是对的，因为 s1 += 1 与 s1 = (short)(s1 + 1)等效,其中隐含了强制类型转换.

```
Integer f1 = 100, f2 = 100, f3 = 150, f4 = 150;

System.out.println(f1 == f2); //true
System.out.println(f3 == f4);//flase
```

如果整型字面量在-128~127之间，那么不会new新的Integer对象，而是直接引用常量池中的Integer对象。所有上面的试题f1 == f2 为true, f3 == f4为false。

```
private static class IntegerCache {
    static final int low = -128;
    static final int high;
    static final Integer cache[];

    static {
        // high value may be configured by property
        int h = 127;
        String integerCacheHighPropValue =
            sun.misc.VM.getSavedProperty("java.lang.Integer.IntegerCache.high");
        if (integerCacheHighPropValue != null) {
            try {
                int i = parseInt(integerCacheHighPropValue);
                i = Math.max(i, 127);
                // Maximum array size is Integer.MAX_VALUE
                h = Math.min(i, Integer.MAX_VALUE - (-low) -1);
            } catch( NumberFormatException nfe) {
                // If the property cannot be parsed into an int, ignore it.
            }
        }
        high = h;

        cache = new Integer[(high - low) + 1];
        int j = low;
        for(int k = 0; k < cache.length; k++)
```

```
            cache[k] = new Integer(j++);

        // range [-128, 127] must be interned (JLS7 5.1.7)
        assert IntegerCache.high >= 127;
    }

    private IntegerCache() {}
}
```

```
public static Integer valueOf(int i) {
    if (i >= IntegerCache.low && i <= IntegerCache.high)
        return IntegerCache.cache[i + (-IntegerCache.low)];
    return new Integer(i);
}
```

HashMap对象的key、value均可以为null，HashTable的key、value均不可以为null

反射

1. 说说你对 Java 中反射的理解

Java 中的反射首先是能够获取到 Java 中要反射类的字节码，获取字节码有三种方法，1.Class.forName(className) 2.类名.class 3.this.getClass()。然后将字节码中的方法，变量，构造函数等映射成相应的 Method、Filed、Constructor 等类，这些类提供了丰富的方法可以被我们所使用。

Java中的基本类型：byte、short、int、long、float、double、char、boolean。存在于jvm的栈中

lambda表达式的运用

1. 可以用distinct( )方法去重：

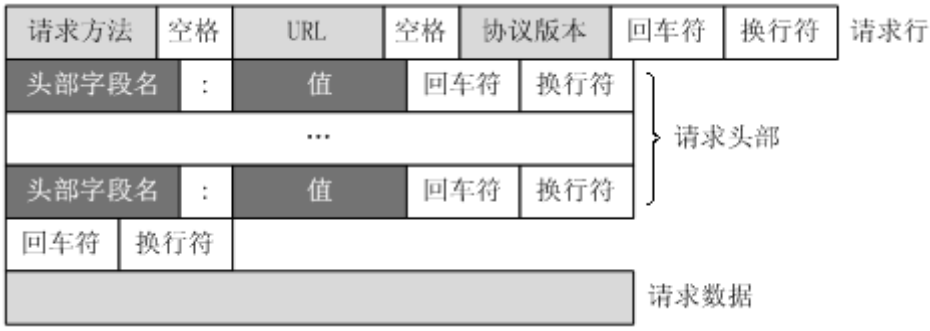
```
List<Integer> list = Arrays.asList(1,2,5,4,3,2,2,6,7,4);
list.stream().distinct().forEach(s -> System.out.print(s + " "));
//1 2 5 4 3 6 7
Stream.of(1,2,3,4,5,6).forEach(System.out::println);
```

2. 计算集合的最大值、最小值、求和以及平均值：

```
List<Integer> list = Arrays.asList(2,3,5,7,10,8,12,20);
IntSummaryStatistics statis = list.stream().mapToInt(x ->
x).summaryStatistics();
System.out.println("最大值:" + statis.getMax());
System.out.println("最小值:" + statis.getMin());
System.out.println("和:" + statis.getSum());
System.out.println("平均值:" + statis.getAverage());
//          最大值:20
//          最小值:2
//          和:67
//          平均值:8.375
```

http请求

一个HTTP请求报文由请求行（request line）、请求头部（headers）、空行（blank line）和请求数据（request body）4个部分组成。



3. http 常见的状态码有哪些？（2017-11-23-wzz）

200 OK　　//客户端请求成功

301 Moved Permanently（永久移除），请求的 URL 已移走。Response 中应该包含一个 Location URL，说明资源现在所处的位置

302 found 重定向

400 Bad Request　//客户端请求有语法错误，不能被服务器所理解

401 Unauthorized //请求未经授权，这个状态代码必须和 WWW-Authenticate 报头域一起使用

403 Forbidden　//服务器收到请求，但是拒绝提供服务

404 Not Found　//请求资源不存在，eg：输入了错误的 URL

500 Internal Server Error //服务器发生不可预期的错误

503 Server Unavailable　//服务器当前不能处理客户端的请求，一段时间后可能恢复正常

1B=8bit：一个字节占8位

1KB=1024B

1MB=1024KB

1GB=1024MB

1TB=1024GB

GET和POST的区别：

1.GET请求的数据会附在URL之后（就是把数据放置在HTTP协议头中），以?分割URL和传输数据，参数之间以&相连，如：login.action?name=hyddd&password=idontknow&verify=%E4%BD%A0%E5%A5%BD。如果数据是英文字母/数字，原样发送，如果是空格，转换为+，如果是中文/其他字符，则直接把字符串用BASE64加密，得出如：%E4%BD%A0%E5%A5%BD，其中%XX中的XX为该符号以16进制表示的ASCII。

POST把提交的数据则放置在是HTTP包的包体中。

2."GET方式提交的数据最多只能是1024字节，理论上POST没有限制，可传较大量的数据，IIS4中最大为80KB，IIS5中为100KB"？？！

以上这句是我从其他文章转过来的，其实这样说是错误的，不准确的：

(1).首先是"GET方式提交的数据最多只能是1024字节"，因为GET是通过URL提交数据，那么GET可提交的数据量就跟URL的长度有直接关系了。而实际上，URL不存在参数上限的问题，HTTP协议规范没有对URL长度进行限制。这个限制是特定的浏览器及服务器对它的限制。IE对URL长度的限制是2083字节(2K+35)。对于其他浏览器，如Netscape、FireFox等，理论上没有长度限制，其限制取决于操作系统的支持。

注意这是限制是整个URL长度，而不仅仅是你的参数值数据长度。[见参考资料5]

(2).理论上讲，POST是没有大小限制的，HTTP协议规范也没有进行大小限制，说“POST数据量存在80K/100K的大小限制”是不准确的，POST数据是没有限制的，起限制作用的是服务器的处理程序的处理能力。

对于ASP程序，Request对象处理每个表单域时存在100K的数据长度限制。但如果使用Request.BinaryRead则没有这个限制。

由这个延伸出去，对于IIS 6.0，微软出于安全考虑，加大了限制。我们还需要注意：

1).IIS 6.0默认ASP POST数据量最大为200KB，每个表单域限制是100KB。
2).IIS 6.0默认上传文件的最大大小是4MB。
3).IIS 6.0默认最大请求头是16KB。

IIS 6.0之前没有这些限制。[见参考资料5]

所以上面的80K，100K可能只是默认值而已(注：关于IIS4和IIS5的参数，我还没有确认)，但肯定是可以自己设置的。由于每个版本的IIS对这些参数的默认值都不一样，具体请参考相关的IIS配置文档。

3.在ASP中，服务端获取GET请求参数用Request.QueryString，获取POST请求参数用Request.Form。在JSP中，用request.getParameter("XXXX")来获取，虽然jsp中也有request.getQueryString()方法，但使用起来比较麻烦，比如：传一个test.jsp?name=hyddd&password=hyddd，用request.getQueryString()得到的是：name=hyddd&password=hyddd。在PHP中，可以用$*GET和*$POST分别获取GET和POST中的数据，而$*REQUEST则可以获取GET和POST两种请求中的数据。值得注意的是，JSP中使用request和PHP中使用*$REQUEST都会有隐患，

4.POST的安全性要比GET的安全性高。注意：这里所说的安全性和上面GET提到的“安全”不是同个概念。上面“安全”的含义仅仅是不作数据修改，而这里安全的含义是真正的Security的含义，比如：通过GET提交数据，用户名和密码将明文出现在URL上，因为(1)登录页面有可能被浏览器缓存，(2)其他人查看浏览器的历史纪录，那么别人就可以拿到你的账号和密码了，除此之外，使用GET提交数据还可能会造成Cross-site request forgery攻击。

总结一下，Get是向服务器发索取数据的一种请求，而Post是向服务器提交数据的一种请求，在FORM（表单）中，Method默认为"GET"，实质上，GET和POST只是发送机制不同，并不是一个取一个发！

5. http 中重定向和请求转发的区别？（2017-11-23-wzz）

本质区别：转发是服务器行为，重定向是客户端行为。

重定向特点：两次请求，浏览器地址发生变化，可以访问自己 web 之外的资源，传输的数据会丢失。

请求转发特点：一次强求，浏览器地址不变，访问的是自己本身的 web 资源，传输的数据不会丢失。

cookie和session的区别：

1、cookie数据存放在客户的浏览器上，session数据放在服务器上.

2、cookie不是很安全，别人可以分析存放在本地的COOKIE并进行COOKIE欺骗考虑到安全应当使用session。

3、设置cookie时间可以使cookie过期。但是使用session-destory（），我们将会销毁会话。
4、session会在一定时间内保存在服务器上。当访问增多，会比较占用你服务器的性能考虑到减轻服务器性能方面，应当使用cookie。
5、单个cookie保存的数据不能超过4K，很多浏览器都限制一个站点最多保存20个cookie。(Session对象没有对存储的数据量的限制，其中可以保存更为复杂的数据类型)

注意:

session很容易失效,用户体验很差;

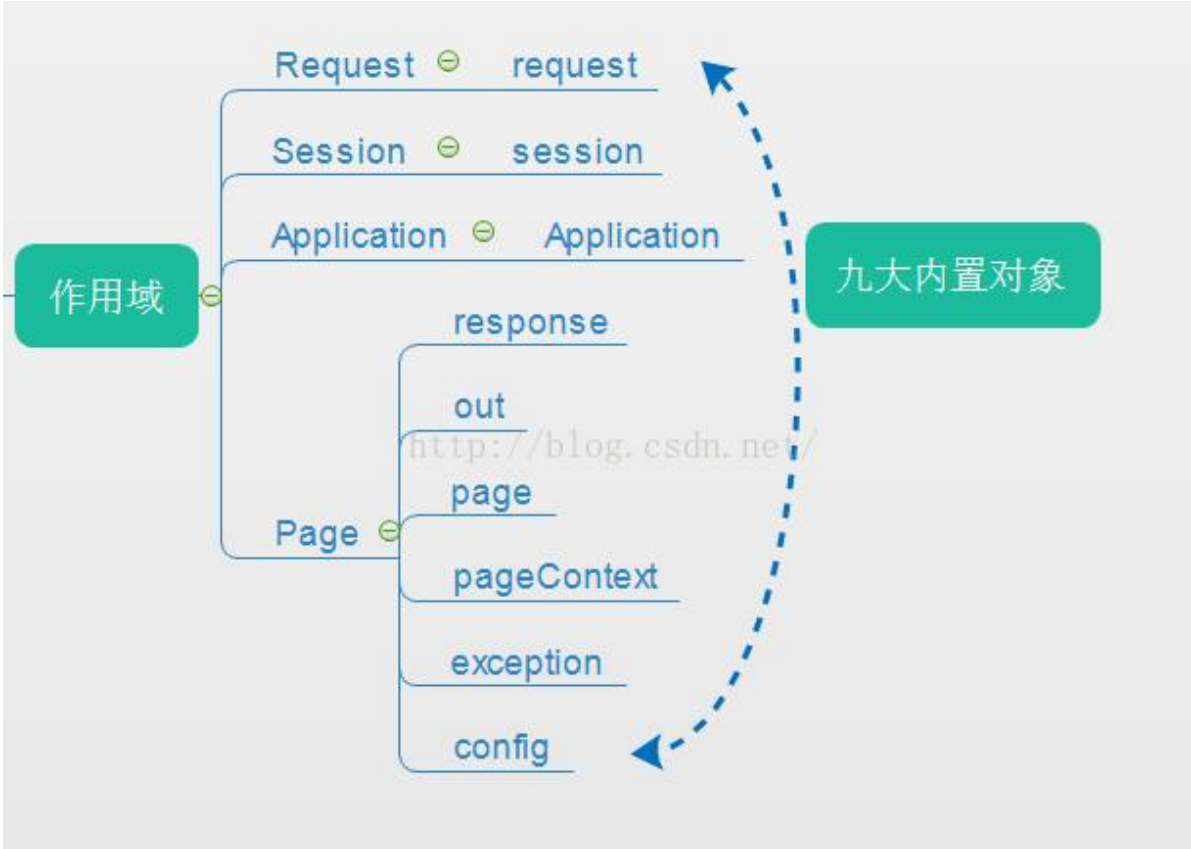
虽然cookie不安全,但是可以加密 ;

cookie也分为永久和暂时存在的;

浏览器 有禁止cookie功能 ,但一般用户都不会设置;

一定要设置失效时间,要不然浏览器关闭就消失了;

jsp四大域和九大内置对象对象：



可以这么记：

9大内置对象对应的类：

内置对象名	类型
request	HttpServletRequest
response	HttpServletResponse
config	ServletConfig
application	ServletContext
session	HttpSession
exception	Throwable
page	Object(this)
out	JspWriter
pageContext	PageContext

4大域作用范围:

```
page域:    只能在当前jsp页面使用                    （当前页面）
request域: 只能在同一个请求中使用                   （转发）
session域: 只能在同一个会话(session对象)中使用  （私有的）
context域: 只能在同一个web应用中使用                （全局的）
```

xml技术：

什么是xml：xml是一种可扩展性标记语言，支持自定义标签(使用前必须预定义)使用DTD和XML Schema标准化XML结构。

xml常用解析器：DOM、SAX

类加载的执行过程

加载一个类，首先会加载其父类

- 父类静态成员变量和父类静态代码块
- 子类静态成员变量和子类静态代码块
- 父类非静态成员变量和非静态代码块
- 父类构造函数
- 子类非静态成员变量和非静态代码块
- 子类构造函数

抽象类和接口有什么异同

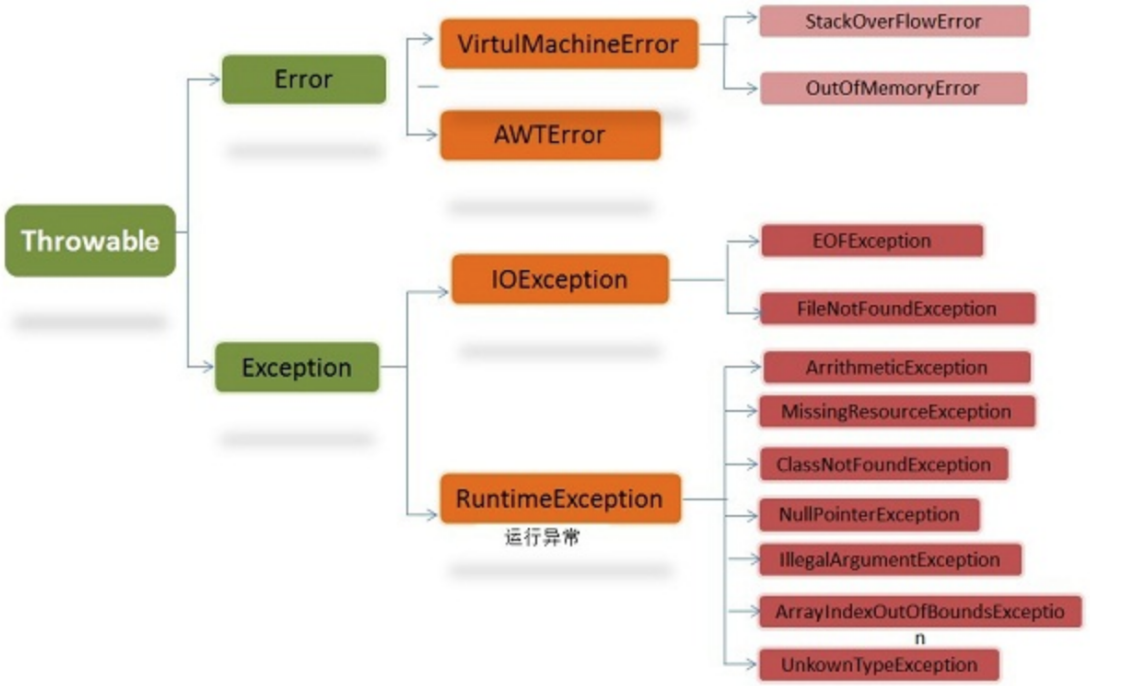
区别：

1. 抽象类可以提供成员方法的实现细节，而接口中只能存在public abstract 方法；
2. 抽象类中的成员变量可以是各种类型的，而接口中的成员变量只能是public static final类型的；
3. 接口中不能含有静态代码块以及静态方法，而抽象类可以有静态代码块和静态方法；
4. 一个类只能继承一个抽象类，而一个类却可以实现多个接口。

相同:

1. 都可以被继承
2. 都不能被实例化
3. 都可以包含方法声明
4. 派生类必须实现未实现的方法

异常



常见的Runtime Exception：

NullPointerException - 空指针引用异常
ClassCastException - 类型强制转换异常。
IllegalArgumentException - 传递非法参数异常。
ArithmeticException - 算术运算异常
ArrayStoreException - 向数组中存放与声明类型不兼容对象异常
IndexOutOfBoundsException - 下标越界异常
NegativeArraySizeException - 创建一个大小为负数的数组错误异常
NumberFormatException - 数字格式异常
SecurityException - 安全异常
UnsupportedOperationException - 不支持的操作异常

SQLException：操作数据库异常类

线程池

《阿里巴巴Java开发手册》中强制线程池不允许使用 Executors 去创建，而是通过 ThreadPoolExecutor 的方式，这样的处理方式让写的同学更加明确线程池的运行规则，规避资源耗尽的风险

> Executors 返回线程池对象的弊端如下：
>
> - FixedThreadPool 和 SingleThreadExecutor： 允许请求的队列长度为 Integer.MAX_VALUE,可能堆积
> 大量的请求，从而导致OOM。
> - CachedThreadPool 和 ScheduledThreadPool： 允许创建的线程数量为 Integer.MAX_VALUE ， 可能
> 会创建大量线程，从而导致OOM。

创建线程池的两种方法:

1. new ThreadPool()构造函数的方式

ThreadPoolExecutor(int, int, long, TimeUnit, BlockingQueue<Runnable>)
ThreadPoolExecutor(int, int, long, TimeUnit, BlockingQueue<Runnable>, ThreadFactory)
ThreadPoolExecutor(int, int, long, TimeUnit, BlockingQueue<Runnable>, RejectedExecutionHandler)
ThreadPoolExecutor(int, int, long, TimeUnit, BlockingQueue<Runnable>, ThreadFactory, RejectedExecutionHandler)

2. Executors工具类

Executors()
newFixedThreadPool(int): ExecutorService
newWorkStealingPool(int): ExecutorService
newWorkStealingPool(): ExecutorService
newFixedThreadPool(int, ThreadFactory): ExecutorService
newSingleThreadExecutor(): ExecutorService
newSingleThreadExecutor(ThreadFactory): ExecutorService
newCachedThreadPool(): ExecutorService
newCachedThreadPool(ThreadFactory): ExecutorService
newSingleThreadScheduledExecutor(): ScheduledExecutorService
newSingleThreadScheduledExecutor(ThreadFactory): ScheduledExecutorService
newScheduledThreadPool(int): ScheduledExecutorService
newScheduledThreadPool(int, ThreadFactory): ScheduledExecutorService

类什么时候被初始化

1. 创建类的实例，也就是new一个对象
2. 访问一个类或接口的静态变量，或对改静态变量赋值
3. 调用类的静态方法
4. 反射Class.forName。。。。三种
5. 初始化一个类的子类（首先会初始化其父类）
6. jvm启动时表明的启动类

原生jdbc操作数据库流程

1. 注册驱动程序：Class.forName(“com.mysql.jdbc.Driver”);
2. 使用驱动管理类来获取数据连接对象：conn = DriverManager.getConnection(...);
3. 获取数据库操作对象：Statement stmt = conn.createStatement()或获取PrepareStatement
4. 定义操作的SQL语句
5. 执行SQL：stmt.executeQuery(sql);
6. 处理结果集：ResultSet，如果SQL前有参数值就设置参数值setXXX()
7. 关闭对象，回收数据库资源（关闭结果集->关闭数据库操作对象->关闭连接）

```
public class JDBCTest {
    /**
     * 使用JDBC连接并操作mysql数据库
     */
    public static void main(String[] args) {
        // 数据库驱动类名的字符串
        String driver = "com.mysql.jdbc.Driver";
        // 数据库连接串
        String url = "jdbc:mysql://127.0.0.1:3306/jdbctest";
        // 用户名
        String username = "root";
        // 密码
        String password = "1234";
        Connection conn = null;
        Statement stmt = null;
        ResultSet rs = null;
        try {
            // 1、加载数据库驱动（ 成功加载后，会将Driver类的实例注册到DriverManager类中）
```

```
            Class.forName(driver);
            // 2、获取数据库连接
            conn = DriverManager.getConnection(url, username, password);
            // 3、获取数据库操作对象
            stmt = conn.createStatement();
            // 4、定义操作的SQL语句
            String sql = "select * from user where id = 100";
            // 5、执行数据库操作
            rs = stmt.executeQuery(sql);
            // 6、获取并操作结果集
            while (rs.next()) {
                System.out.println(rs.getInt("id"));
                System.out.println(rs.getString("name"));
            }
        } catch (Exception e) {
            e.printStackTrace();
        } finally {
            // 7、关闭对象，回收数据库资源
            if (rs != null) { //关闭结果集对象
                try {
                    rs.close();
                } catch (SQLException e) {
                    e.printStackTrace();
                }
            }
            if (stmt != null) { // 关闭数据库操作对象
                try {
                    stmt.close();
                } catch (SQLException e) {
                    e.printStackTrace();
                }
            }
            if (conn != null) { // 关闭数据库连接对象
                try {
                    if (!conn.isClosed()) {
                        conn.close();
                    }
                } catch (SQLException e) {
                    e.printStackTrace();
                }
            }
        }
    }
}
```

使用PreparedStatement的方式：

```
public void testPreparedStatement() {
        Connection connection = null;
        PreparedStatement preparedStatement = null;
        try {
            // 连接数据库
            connection = JDBCTools.getConnection();
            // 使用占位符的SQl语句
            String sql = "insert into customers(name,email,birth)"
                    + "values(?,?,?)";
            // 使用preparedStatement的setXxx方法设置每一个位置上的值
```

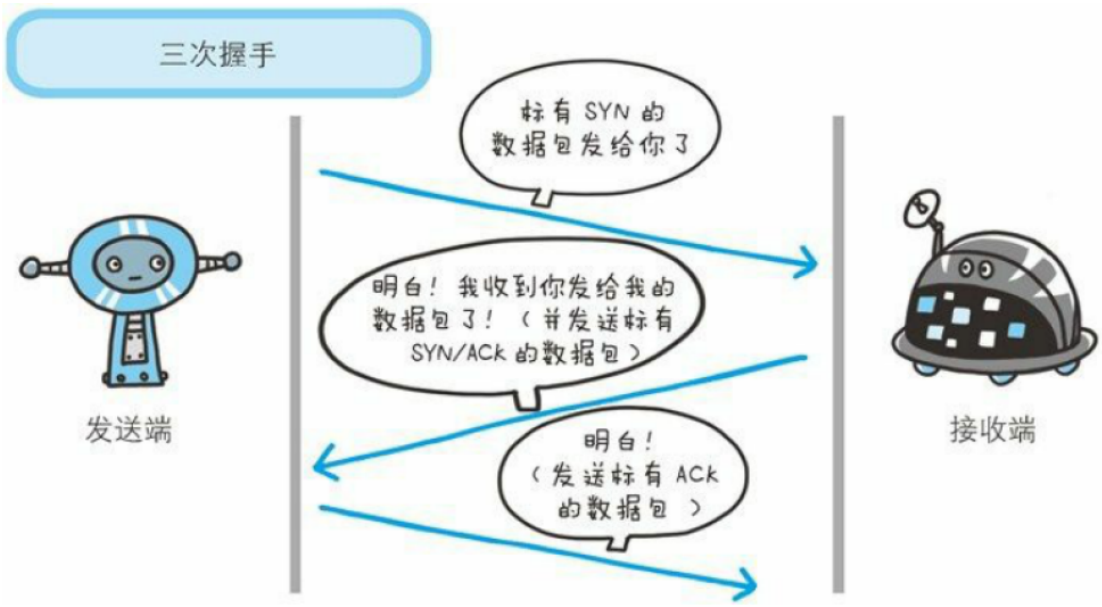
```
            preparedStatement = connection.prepareStatement(sql);
            // 设置name字段
            preparedStatement.setString(1, "ATGUIGU");
            // 设置email字段
            preparedStatement.setString(2, "simale@163.com");
            // 设置birth字段
            preparedStatement.setDate(3,
                    new Date(new java.util.Date().getTime()));
            // 执行更新操作
            preparedStatement.executeUpdate();
        } catch (Exception e) {
            e.printStackTrace();
        } finally {
            // 释放资源
            JDBCTools.release(null, preparedStatement, connection);
        }
    }
```

```
public void testSQLinjection2() {
    String username = "a' or password =";
    String password = " or '1'='1";
    String sql = "select * from users where username=?" + " and password=?";
    System.out.println(sql);
    Connection connection = null;
    PreparedStatement preparedStatement = null;
    ResultSet resultSet = null;
    try {
        connection = getConnection();
        preparedStatement = connection.prepareStatement(sql);
        preparedStatement.setString(1, username);
        preparedStatement.setString(2, password);
        resultSet = preparedStatement.executeQuery();
        if (resultSet.next()) {
            System.out.println("登陆成功");
        } else {
            System.out.println("不匹配");
        }
    } catch (Exception e) {
        e.printStackTrace();
    } finally {
        JDBCTools.release(resultSet, preparedStatement, connection);
    }
}
```

使用PreparedStatement的好处：

1. 提高代码的可读性和可维护性，PreparedStatement接口继承Statement，PreparedStatement实例以包含编译的SQL语句，执行速度要快于Statement
2. 最大程度的提高性能，作为Statement的子类，PreparedStatement继承了Statement的所有功能。execute、executeQuery、executeUpdate已被修改，使之不需要参数
3. 防止SQL注入，PreparedStatement传入的内容不会和SQL发生任何的匹配关系

HTTP三次握手



为什么要三次握手：

第一次握手：Client 什么都不能确认；Server 确认了对方发送正常
第二次握手：Client 确认了：自己发送、接收正常，对方发送、接收正常；Server 确认了：自己接收正常，对方发送正常
第三次握手：Client 确认了：自己发送、接收正常，对方发送、接收正常；Server 确认了：自己发送、接收正常，对方发送、接收正常
所以三次握手就能确认双发收发功能都正常，缺一不可。

JDK和JRE的区别

从图中可以看出JDK包含JRE包含JVM...

JDK：java development kit （java开发工具包）

JRE：java runtime environment （java运行时环境）

JVM：java virtuak machine （java虚拟机）

客户端禁止 cookie，session 还能用吗

一般情况下，客户端的cookie里面存有sessionId，服务器通过客户端里的cookie获取sessionId，再查询服务器里的session。

如果浏览器禁用了 cookie，浏览器请求服务器无法携带 sessionid，服务器无法识别请求中的用户身份，session失效。但是可以通过其他方法在禁用 cookie 的情况下，可以继续使用session：

1. 通过url重写，把 sessionid 作为参数追加的原 url 中，后续的浏览器与服务器交互中携带 sessionid 参数。
2. 服务器的返回数据中包含 sessionid，浏览器发送请求时，携带 sessionid 参数。
3. 通过 Http 协议其他 header 字段，服务器每次返回时设置该 header 字段信息，浏览器中 js 读取该 header 字段，请求服务器时，js设置携带该 header 字段。