Java6、7、8

java的历史。过去，现在，未来

java的历史

Sun决定开发一种集C语言和Mesa语言大成的新语言，在一份报告上，Sun把它叫做“未来”，Sun在C++的基础上，开发一种面向对象的环境。最初，高斯林试图修改和扩展C++的功能，他自己称这种新语言为C++ ++ --，但是后来他放弃了。他将要创造出一种全新的语言，被他命名为“Oak”(橡树)，以他的办公室外的橡树命名。

C++的问题和java的流行

C++:

无可否认C++的强大。或者说是过于强大。

传统的面向过程的编程和面向对象的编程。以及从C继承下来的底层操作让C++显得非常非常的强大。

但C++有问题。

或许不是C++的问题。是使用C++的人的问题。但是

C++确实不是适合大规模工业化的语言

不是人人都是科学家，数学家。编程爱好者。总有人得干点脏活累活。

------论Java的诞生

Java对于C++暴露出来的问题。根本没有解决。而是阉割掉了

C++的问题和java的办法

问题	C++	java
指针和垃圾回收	有。自己操控。C++有智能指针的概念，优点。对程序更好的掌控	java没有指针。垃圾自主回收 指针是个麻烦的东西。你根本就不知道哪里出 错了
Macro	C支持的东西继承。不过我们提供了inline	为什么需要宏
继承	我们提供了灵活的继承，但是好像出 现了重复继承。多继承好像又很复 杂。但是我们只能加个关 键字virtual	单继承。 接口的话直接interface把
包管理	#include 好像还是有冲突。那就namespace吧	import。域名不可能重复

问题	C++	java
编译运行	不同平台不同 编译。不同维护	一次编译一次运行
基本类	除了类还有很多	除了Int/Double一切都是对象
错误	Exception没有的话。会暴露到Stack。但是不会告 诉是什么原因	有到完整的NullException到各种各样的Exception类

java的兴起

互联网：

一次编译多处运行。（实际上：一次编译到处Debug）

大规模的开源项目。Java良好的规则和语法适合开源项目。

Java 5,6

1.支持Annotation。整个Spring基础发展的基础。切面的基础

2,JDBC4.0支持

java7

switch中可以使用字串了

```
string s = "test";
switch (s) {
case "test" :
system.out.println("test");
break ;
default :
system.out.println("break");
break ;
}
```

安全計算包:
long Math.safeSubtract(long value1, int value2)

java8

默认接口方法：

Java 8允许我们给接口添加一个非抽象的方法实现，只需要使用*default*关键字即可，这个特征又叫做扩展方法，示例如下：

```
interface Formula {
  double calculate(int a);
   default double sqrt(int a) {
      return Math.sqrt(a);
}}

Formula formula = new Formula() {
@Override
 public double calculate(int a) {
   return sqrt(a * 100);    }
                    };
```

Lambada

匿名类

```
List<String> names = Arrays.asList("peter",
"anna", "mike", "xenia");
Collections.sort(names, new
Comparator<String>() {
    @Override
    public int compare(String a, String b) {
        return b.compareTo(a);
    }
});
```

基本上必须只有一个接口
多个以上就必须加入@functionInterface

lambada：

```
Collections.sort(names, (String a, String b) -> {
    return b.compareTo(a);
});
Collections.sort(names, (String a, String b) ->
b.compareTo(a));
```

```
Collections.sort(names, (a, b) -> b.compareTo
(a));
```

这是给大Scala给跪的节奏

lambada 可访问范围

```
int num = 1;
Converter<Integer, String> stringConverter =
(from) -> String.valueOf(from + num);
stringConverter.convert(2);     // 3
```

可以访问外面的变量。关键点是num是final

在改变会出错

interface自动推导静态类

```
public class Hello {
   public static void main(String[] args)
   {
      String something="tis is";
      Converter<String, Boolean> converter = something::startsWith;
      Converter<String,Integer> converter2=Interger::valueof
      Boolean converted = converter.convert("1");
      Integer a=converter2.convert(”2“);//2
      System.out.println(converted);   // true
   }

    @FunctionalInterface
    interface Converter<F, T> {
        T convert(F from);
    }
}
```

New Data api

Clock类提供了访问当前日期和时间的方法，Clock是时区敏感的，可以用来取代 System.currentTimeMillis() 来获取当前的微秒数。某一个特定的时间点也可以使用Instant类来表示，Instant类也可以用来创建老的java.util.Date对象。

```
Clock clock = Clock.systemDefaultZone();
long millis = clock.millis();
Instant instant = clock.instant();
Date legacyDate = Date.from(instant);   // legacy java.util.Date
```

Timezones 时区

在新API中时区使用ZoneId来表示。时区可以很方便的使用静态方法of来获取到。时区定义了到UTS时间的时间差，在Instant时间点对象到本地日期对象之间转换的时候是极其重要的。

```
System.out.println(ZoneId.
getAvailableZoneIds());
ZoneId zone1 = ZoneId.of("Europe/Berlin");
ZoneId zone2 = ZoneId.of("Brazil/East");
System.out.println(zone1.getRules());
System.out.println(zone2.getRules());
                                                    //
ZoneRules[currentStandardOffset=+01:00]
                                                    //
ZoneRules[currentStandardOffset=-03:00]
```

LocalDate

LocalDate 表示了一个确切的日期, 比如 2014-03-11。该对象值是不可变的, 用起来和LocalTime基本一致。下面的例子展示了如何给Date对象加减天/月/年。另外要注意的是这些对象是不可变的, 操作返回的总是一个新实例。

```
LocalDate today = LocalDate.now();
LocalDate tomorrow = today.plus(1, ChronoUnit.DAYS);
LocalDate yesterday = tomorrow.minusDays(2);
LocalDate independenceDay = LocalDate.of(2014, Month.JULY, 4);
DayOfWeek dayOfWeek = independenceDay.getDayOfWeek();
System.out.println(dayOfWeek);    // FRIDAY
```

Annotation的注解

注解可以再加个注解

```
@interface Hints {
    Hint[] value();
}

@Repeatable(Hints.class)
@interface Hint {
    String value();
}
```

以前用法:

```
@Hints({@Hint("hint1"), @Hint("hint2")})
class Person {}
```

现在用法:

```
@Hint("hint1")
@Hint("hint2")
class Person {}
```