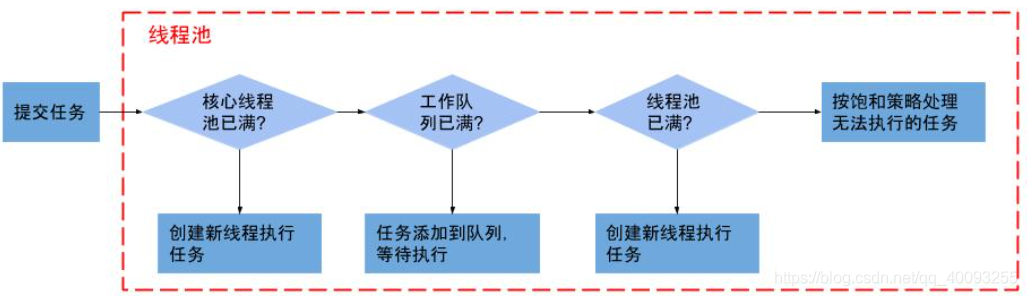
## 线程池：

### 参数：

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| ***工厂方法*** | **corePoolSize** | **maximumPoolSize** | **keepAliveTime** | **workQueue** |
| *newCachedThreadPool* | 0 | Integer.MAX\_VALUE | 60s | SynchronousQueue |
| *newFixedThreadPool* | nThreads | nThreads | 0 | LinkedBlockingQueue |
| *newSingleThreadExecutor* | 1 | 1 | 0 | LinkedBlockingQueue |
| *newScheduledThreadPool* | corePoolSize | Integer.MAX\_VALUE | 0 | DelayedWorkQueue |

### 流程：



### 基本使用：

*ExecutorService executor = Executors.newFixedThreadPool(5);*

*executor.submit(()->{});*

### 特点：

newCachedThreadPool

* 可缓存

newFixedThreadPool：

* 一个指定大小的线程池

newScheduledThreadPool：

* 支持定时及周期性任务的执行

newSingleThreadExecutor：

* 只有一个线程

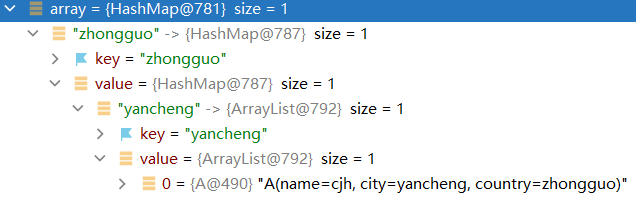
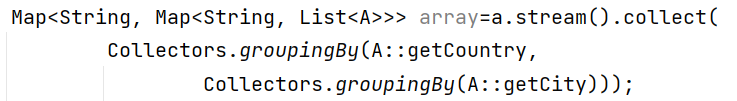
### 线程工厂类：

|  |
| --- |
| *public ThreadPoolExecutor(int corePoolSize,*  *int maximumPoolSize,*  *long keepAliveTime,*  *TimeUnit unit,*  *BlockingQueue<Runnable> workQueue){*  *this(corePoolSize, maximumPoolSize, keepAliveTime, unit, workQueue,Executors.defaultThreadFactory(), defaultHandler);*  *}*  ***/\****  *unit（存活时间的单位）*  *workQueue（任务队列）不能为空*  *threadFactory（线程工厂）不能为空，默认为DefaultThreadFactory类*  *handler（线程饱和策略）不能为空，默认策略为ThreadPoolExecutor.AbortPolicy*   * *ThreadPoolExecutor.AbortPolicy：处理程序遭到拒绝，则直接抛出运行时异常 RejectedExecutionException。(默认策略)* * *ThreadPoolExecutor.CallerRunsPolicy：调用者所在线程来运行该任务，此策略提供简单的反馈控制机制，能够减缓新任务的提交速度。* * *ThreadPoolExecutor.DiscardPolicy：无法执行的任务将被删除。* * *ThreadPoolExecutor.DiscardOldestPolicy：如果执行程序尚未关闭，则位于工作队列头部的任务将被删除，然后重新尝试执行任务（如果再次失败，则重复此过程）。*   ***\*/*** |

## Stream操作：

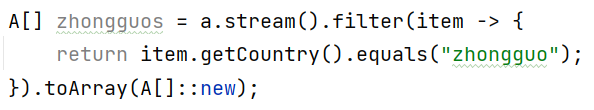
### collect():

可以对流进行收集到容器里、（多级）分组、（多级）分区、拼接、聚合等操作



### filter():

可以对流进行筛选



### peek()、map()、foreach():

peek() 参数为无返回值的方法

map() 参数为有返回值的方法

foreach() 会终止流

### limit(int):

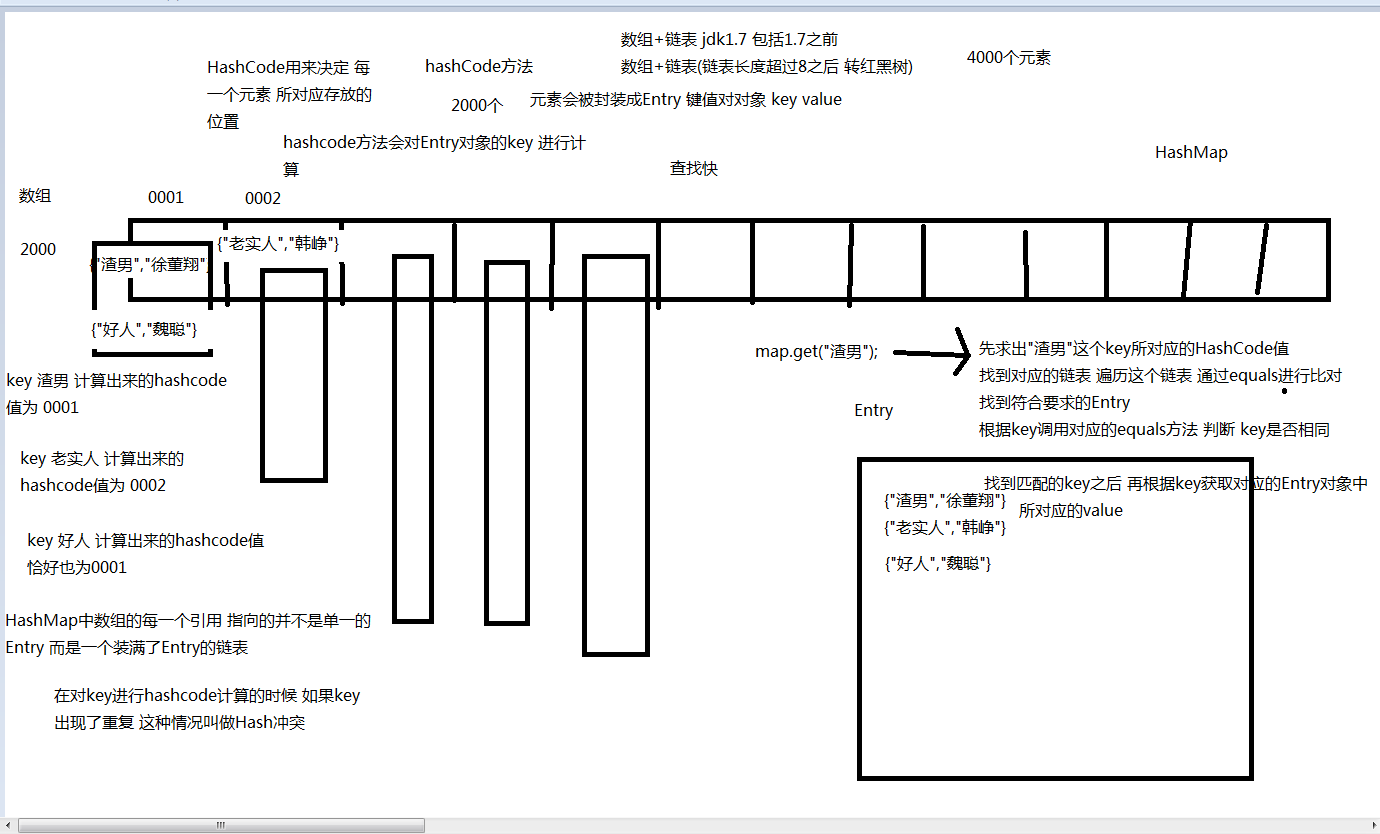
截取流，返回

## 容器 数据结构：

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| M  a  p | Hash  Map | Linked  Hash  Map | **ArraryList<**  **List<**  **Entry<**  **K,V>>>**  Map.  Entry  <K,V> | clear()  remove()  put()  putAll()  get()  containsKey() containsValue() |
| HashTable  (线程安全) | |
| TreeMap | |
| equals()  hashCode() |
| entrySet()  keySet()  values() |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Co  ll  ec  ti  on  (集合  接  口) | **List接口（有索引）（可重复）** | **ArraryList**  **（数组实现）**  **（有序）** | | add()  remove()  contains()  addAll  removeAll  containsAll  retainAll  equals()  hashcode() | 查询修改快 |
| **LinkedList（链表实现）**  **（有序）** | | 增删快 |
| **(淘汰)Vector（数组实现）（有序）** | |  |
| **Set接口（无索引）（不可重复）** | **Hash**  **Set**  **（哈希算法）（无序）** | **Linked**  **Hash**  **Set**  **（链表实现）**  **（有序）** | 由HashMap封装 |
| **TreeSet（二叉树算法）**  **（无序）** | |  |

### HashMap结构



### Collections

Collections.sort(List<> a,Comparator <>b)

## 类加载顺序

### 结论:

|  |  |
| --- | --- |
| 未实例化时 | 实例化时 |
| 1. 加载父类 | 1.先进行未实例化时的操作 |
| 2.从上往下加载static块 | 2.父类成员(子类构造函数默认调用子类.super()导致) |
| 3.返回子类, 从上往下加载static块 | 3.父类构造函数 |
| static块只加载一次 | 4.子类成员 |
| 如果static中有实例化操作，转到实例化时 | 5.子类构造函数 |

### 测试用例:

|  |
| --- |
| public class B21 {  private int a;  protected int b;  static int c=10;  B21(){  System.out.println("B21 第一");  }  static{  System.out.println("B21 第二");  }  static void d(){  System.out.println("B21 第三");  }  B21(int a){  System.out.println("B21 第四");  }  void e(){  System.out.println("B21 第五");  }  } |
| public class B21\_1 extends B21{  private int aa;  protected int bb;  static int cc=10;  B21\_2 f=new B21\_2();  static{  System.out.println("B21\_1 第二");  }  B21 ff=new B21(2);  static B21 gg=new B21(2);  B21\_1(){  System.out.println("B21\_1 第一");  }  static void dd(){  System.out.println("B21\_1 第三");  }  B21\_1(int a){  super(2);  System.out.println("B21\_1 第四");  }  void ee(){  System.out.println("B21\_1 第五");  }  public static void main(String[] args) {  B21\_1 a=new B21\_1(2);  }  } |
| public class B21\_2 {  B21\_2(){  System.out.println("B21\_2");  }  } |

## lambda

### 结论：

1.lambda用来替代匿名内部类，实现代码的简洁

1）内部接口是内部类，因此可以用lambda代替，但是这个接口只能有一个方法，是为函数式编程

2) 外部接口同理

### 测试用例:

|  |
| --- |
| public class B20 {  /\* 内部接口  public interface B20\_addable{  public int add(int a,int b);  } \*/  // 1.法一：直接实例化接口  public int do\_addable\_1(int mm,int nn){  /\*addable a=new addable() {  @Override  public int add(int a, int b) {  return a+b;  }  }; 用lambda改写如下\*/  B20\_addable a=(m,n)->m+n;  return a.add(mm,nn);  }  //法一  //法二：定义新类包含接口  public class B20\_1 implements B20\_addable{  @Override  public int add(int m,int n){  return m+n;  }  };  public int do\_addable\_2(int mm,int nn){  return new B20\_1().add(mm,nn);  }  //法二  public static void main(String[] args) {  B20 b20=new B20();  System.out.println(b20.do\_addable\_1(2,1));  System.out.println(b20.do\_addable\_2(2,1));  }  } |

## 基本数据类型的封装类

byte Byte

boolean Boolean

char Character

short Short

int Integer

long Long

float Float

double Double

### 装箱 拆箱

## String

|  |
| --- |
| String a = "abc";  String b = new String("abc");  String c = a + b;//堆中新创建了两个对象  //等价于new StringBuilder().append(a).append(b).toString(); |

String 本身是被final修饰,不可以被继承,不可以被修改的

format() 使用指定的格式字符串和参数返回格式化字符串。

|  |
| --- |
| String strFormat ="%s 在被了连续揍了 %d 次后，大口问到 %s %n";  String str = String.format(sentenceFormat, "赵文宇",5,"你在赣神魔");  System.out.println(str); |

输出任何类型对象，会隐式调用toString()方法，因此可以在自定义类里重写toString()方法

### 常用方法

charAt

获取字符

toCharArray

获取对应的字符数组

subString

截取子字符串

split

分隔

trim

去掉首尾空格

toLowerCase

toUpperCase

大小写

indexOf

lastIndexOf

contains

定位

replaceAll

replaceFirst

startsWith

endsWith

## StringBuffer StringBuilder

可变字符串

性能StringBuilder速度快

安全性StringBuffer安全

append()方法 调用一次只会增加capacity一次。不超过则容量为（旧容量+1）\*2，超过则容量为新StringBuff的长度

### 常用方法

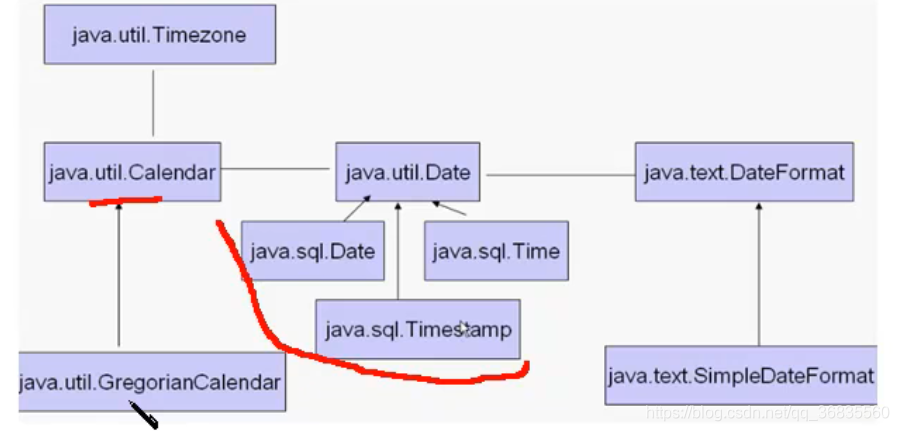
append 追加

delete 删除

insert 插入

reverse 反转

## 时间类



### SimpleDateFormat

构造函数 SmipleDateFormat a=new SimpleDateFormat("yy MM dd HH mm ss")

Date c=a.parse(String b); 按a的格式格式化文本返回Date对象

String d=a.format(c) 按a的格式格式化Date对象

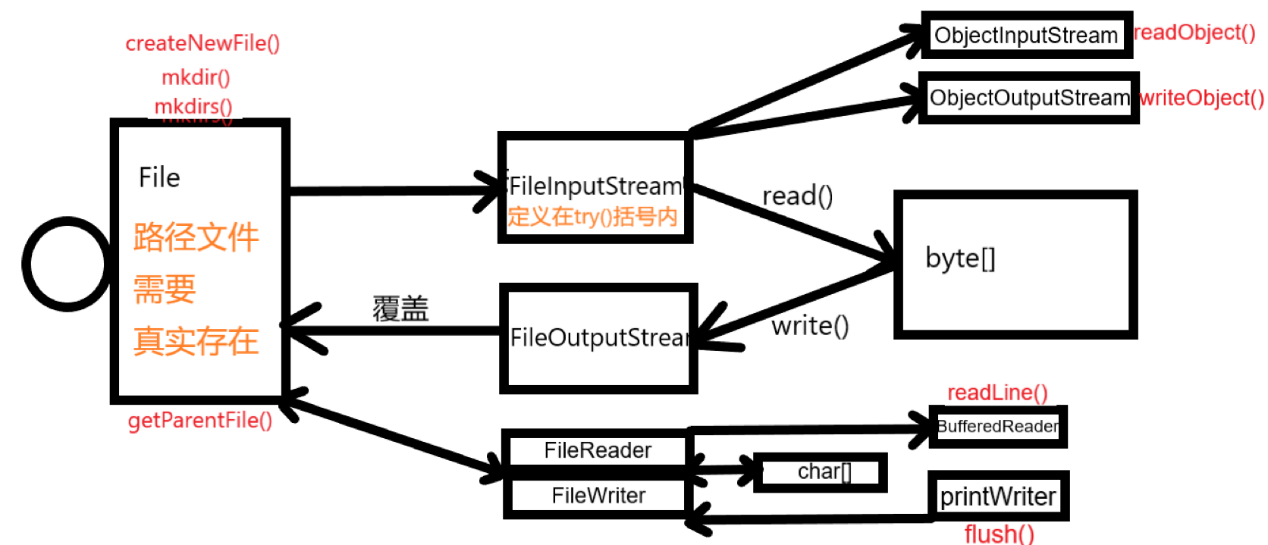
### Calendar类

|  |
| --- |
| public class TestDate {  private static SimpleDateFormat sdf = new SimpleDateFormat("yyyy-MM-dd HH:mm:ss");  public static void main(String[] args) {  Calendar c = Calendar.getInstance();  Date now = c.getTime();  c.add(Calendar.YEAR|MONTH|DATE, 1);  c.set(Calendar.DATE, 3);  }  } |

## 文件类和流

|  |
| --- |
| File f=new File("D:/sdasdsa/sdasdasd/zzzz");  System.out.println(  f.getPath()+"\n"+//绝对路径  f.getAbsoluteFile()+"\n"+//绝对路径  f.exists()+"\n"+//是否存在  f.isFile()+"\n"+//是文件类型  f.isDirectory()+"\n"+//是目录  f.length()+"\n"+//长度  new SimpleDateFormat("yyyy-MM-dd HH:mm:ss").format(f.lastModified())//最后修改日期  );  //f.list();//获取当前目录下所有文件和目录的名字集合  //f.listFiles();//获取当前目录下所有文件和目录的文件对象的集合  //f.mkdir() f.mkdirs()  //f.listRoots() 返回所有盘符文件  //f.delete() jvm退出后删除 |

### 类图



"."表示当前项目根目录

"/"表示当前盘符根目录

## JDBC

|  |
| --- |
| public class A {  public static void main(String[] args) throws Execption{  //加载驱动  Class.forName("com.mysql.cj.jdbc.Driver");  Connection connection=null;  Statement statement=null;  //创建进程 提供要连接的数据库的基本信息  //mysql IP地址 端口号 DB和相关参数  //SSL 安全认证协议  connection = DriverManager.getConnection("jdbc:mysql://120.24.52.179:3306/aba?characterEncoding=utf8&useSSL=false&serverTimezone=UTC", "admin", "123456");  statement=connection.createStatement();  //创建编译语句  String sql=String.format("insert into students values(null,%s,18)","'韩庚'"); //韩庚双引号单引号  --------------------------------------------  statement.execute(sql);//执行编译语句  --------------------------------------------  connection.close();//关闭连接  }  } |

### 查

|  |
| --- |
| public class A {  public static void main(String[] args) throws Exception{  Class.forName("com.mysql.cj.jdbc.Driver");  Connection connection=null;  Statement statement=null;  connection= DriverManager.getConnection("jdbc:mysql://120.24.52.179:3306/aba?characterEncoding=utf8&useSSL=false&serverTimezone=UTC", "admin", "123456");  statement=connection.createStatement();  String order="select \* from students";  --------------------------------------------------  ResultSet resultSet=statement.executeQuery(order);  --------------------------------------------------  while (resultSet.next()){ |------------------------|  int id= resultSet.getInt("id");------------|//如果传入的是下标，从1开始 |  String name=resultSet.getString("name");---|------------------------|  System.out.println(id+" "+name+" ");  }  --------------------------------------------------  connection.close();  }  } |

### PrepareStatement

|  |
| --- |
| String order="select \* from students limit ?,?";  //要先准备sql语句 之后再定义PrepareStatement对象  PreparedStatement preparedStatement=connection.prepareStatement(order);  //用setXXX方法对参数进行设置  preparedStatement.setInt(1,0);  preparedStatement.setInt(2,5); //如果传入的是下标，从1开始 |  //用execute或者executeQuery  ResultSet resultSet=preparedStatement.executeQuery(); |

### 插入 获取自增长键值

|  |
| --- |
| ResultSet resultSet1=preparedStatement.getGeneratedKeys();  ResultSet resultSet1=preparedStatement.executeQuery();  if (resultSet1.next())  System.out.println(resultSet.getInt(1)); |

### 原子性

|  |
| --- |
| //关闭jdbc默认的事务自动提交 关闭后 所有DML操作都需要程序员手动提交 才能把缓存中的数据写入到库中  connection.setAutoCommit(false);  //事务提交  connection.commit();  //事务回滚  connection.rollback(); //在try{}catch(){}中 |

## 线程

run接口

Thread类

线程的插队 join()

线程的让步 yield()

守护线程 setDaemon(true)

synchronized

wait()

notify()

### 线程池

## Socket—NIO（看demo）

|  |
| --- |
| ServerSocket serverSocket=new ServerSocket(8848);  Socket server = serverSocket.accept();  OutputStream os = server.getOutputStream();  DataOutputStream dataOutputStream=new DataOutputStream(os);  InputStream in = server.getInputStream();  DataInputStream dataInputStream=new DataInputStream(in);  ///////////////////////////////////////////////////////////  dataInputStream.close();  in.close();  dataOutputStream.close();  os.close();  server.close(); |

|  |
| --- |
| Socket client = new Socket("127.0.0.1", 8848);  OutputStream os = client.getOutputStream();  DataOutputStream dataOutputStream=new DataOutputStream(os);  InputStream in = client.getInputStream();  DataInputStream dataInputStream=new DataInputStream(in);  /////////////////////////////////////////////////////////  dataInputStream.close();  in.close();  dataOutputStream.close();  os.close();  client.close(); |

## 反射

### 三种获取类的方法

|  |
| --- |
| String s="加载类.A"; //需要完全限定名  Class a=Class.forName(s);  Class b=A.class; //不会加载类  Class c=new A().getClass(); |

### Field类及方法

|  |
| --- |
| Student s=new Student("cjh");  Class class1=s.getClass();  Field f1=class1.getDeclaredField("name");  f1.set(s,"clj"); |

### Constructor类 Method类

|  |
| --- |
| File f=new File("./src/properties.txt");  Properties p=new Properties();  p.load(new FileInputStream(f));  String className=(String)p.get("class");  String methodName1=(String)p.get("methodName1");  String methodName2=(String)p.get("methodName2");  Class class1=Class.forName(className);  Constructor c=class1.getConstructor();  Object class1\_object1=c.newInstance();  Method m1=class1.getMethod(methodName1,String.class);  m1.invoke(class1\_object1,"25");  Method m2=class1.getMethod(methodName2);  System.out.println(m2.invoke(class1\_object1)); |

## ThreadLocal类

### ThreadLocal<T>里面拥有一个ThreadLocalMap的*静态内部类（从依赖关系上，静态内部类和外部类没有任何关系，相当于单独的一个类）*（数据结构和Map类似）。ThreadLocal<T>的set()会为当前线程创建或者更新一个ThreadLocalMap，它的key是这个ThreadLocal，value是程序员自定义的值。弱引用，需要手动remove。

## Https和SSL协议

1. 什么是SSL协议？

SSL(Secure Sockets Layer 安全套接层),及其继任者传输层安全(Transport Layer Security，TLS)是为网络通信提供安全及数据完整性的一种安全协议。

1. 什么是SSL证书？

SSL 证书 就是遵守 SSL协议，由受信任的数字证书颁发机构CA，在验证服务器身份后颁发，具有服务器身份验证和数据传输加密功能。类似于身份证、驾驶证。但是这个证书可以自己制作，因此只有官方认证的证书才是最安全的。有哪些格式？PEM、CER、JKS、PKCS12等等

1. 为什么访问有些https网址需要导入证书？

因为它没购买正版SSL证书，使用的是自生成的证书，此时访问就会报警告。

1. 如何查看证书信息？

可以使用java自带的keytool工具：

keytool -list -v -keystore xxx\server.keystore

其他命令可以参照:<https://blog.csdn.net/liumiaocn/article/details/61921014>

1. 导入证书要配置哪些配置？

Spring boot的application.yml中这样配置：

server:  
 ssl:  
 # 证书路径  
 key-store: classpath:server.keystore  
 key-alias: tomcat  
 enabled: true  
 key-store-type: JKS  
 #与申请时输入一致  
 key-store-password: 123456  
 # 浏览器默认端口 和 80 类似  
 port: 443

## 有返回值的线程

**Callable接口 是执行具体代码的方法  
 FutureTask类 是线程载体，具体干什么由Callable负责（称为线程方法），又继承了Future接口，可以查看线程方法运行情况**

**调用Future或者FutureTask的get方法，会阻塞当前线程的执行，（但不会阻塞其他线程的执行），直到拿到结果。**

**如果FutureTask执行过程中出现异常，不会自动在控制台打印异常信息，原因是FutureTask的run方法会将异常信息保存起来。要么FutureTask.get()获取异常，要么在callable接口中try-catch自行捕获异常。**

## java三大特性

封装（隐藏类的内部细节，只把访问方法暴露出来；属性封装，方法封装；）

继承（子类可以调用父类的属性和方法）

多态（变量只有运行时才能确定其真正的类型）

## Hashmap扩容

初始化要为2的n次方容量。

默认0.75扩容因子，此时链表长度达到8转为红黑树的概率基本为0。