* 今天学习目标

  * 能够掌握自定义异常

    * extends Exception

      * 重写父类

      * UserNameExisitException

  * 能够掌握日志框架log4j和log4j2

    * 级别(trace-->debug-->info-->warn-->error)，目的地，输出格式，配置文件

    * Logger，Appender（ConsoleAppender，FileAppendar，...）,Layout

  * 能够理解集合的概述

    * 容器-->引用类型

  * 能够掌握集合和数组的区别

    * 长度

  * 能够了解集合框架(框架图)

    * Collection--List（ArrayList,LinkedList,..）--Set(HashSet)

  * 能够掌握Collection集合常用的方法

    * add,remove,clear,addAll,removeAll,contains,containsAll,isEmpty，size，toArray

  * 能够使用迭代器Iterator对集合进行遍历

```
Iterator ite=con.iterator();
while(ite.hasNext()){
    ite.next();
}
```

  * 能够使用增强型for对集合进行遍历

```
for(String value:c){


}
```

---

* 回顾

* Runtime

* 单例：

* private sttaic Singleton singleton=new Singleton();

* private Singleton(){}

* public static Singleton getInstance(){

return singleton;

}

* Runtime runtime=Runtime.getRunTime();

* freeMemory(),totalMemory(),maxMemory(),availableProcecosxxx，gc

* exec("mspaint,notepad,...");

* Throwable

* Error --->OutOfMemoryError

* Exception

* RuntimeException(NullPointerException,ArrayOutOfBoundsException,算术,输入不匹配异常，...)

* IOException

* try,catch,finally,throw,throws

* try()catch(){}catch(){},finally{}

* final,finalize,finally

* finally return

```
try{
    int i=1;
    i++;
    return i;
} catch(Exception e){
     i++;
}finally{
```

```
        i=10;
        return i;
    }
```

* 能够掌握自定义异常
    * 注册：用户名已经存在的异常
    * 根据业务需要自定义异常

```
* 注册的时候：检测用户名是否存在
public class RegisterException extends Exception{
    private static final long serialVersionUID = 1L;
    public RegisterException() {

    }
    public RegisterException(String msg) {
        super(msg);
    }
}

public class Test3 {
    public static String[] names= {"xiaohei","xiaobai"};
    public static void main(String[] args) {
        System.out.println("请输入注册的用户名：");
        Scanner input=new Scanner(System.in);
        String name=input.next();
        try {
            checkName(name);
            System.out.println("可以注册...");
        } catch (RegisterException e) {
            e.printStackTrace();
        }
    }
    private static boolean checkName(String name) throws RegisterException {
        for (String userName:names) {
            if(name.equals(userName)) {
                throw new RegisterException("亲,"+name+"已经被注册");
            }
        }
```

```
        return true;
    }
}

结果：
请输入注册的用户名：
xiaohei
com.lg.test.RegisterException: 亲,xiaohei已经被注册
    at com.lg.test.Test3.checkName(Test3.java:21)
    at com.lg.test.Test3.main(Test3.java:12)
```

* 能够掌握日志框架log4j和log4j2

* log4j 是Apache的一个开源项目，通过使用Log4j，我们可以控制日志信息输送的目的地（控制台，文件，服务器），可以控制每一条日志的输出格式，可以定义每一条日志信息的级别，这些可以通过一个配置文件来灵活地进行配置，而不需要修改应用的代码。

* java--class--jar

* 修改配置文件

* Log4j的下载地址：http://logging.apache.org/log4j/1.2/download.html

* Log4j由三个重要的组成构成：日志记录器(Loggers)，输出端(Appenders)和日志格式化器(Layout)

* Logger：控制要启用或禁用哪些日志记录语句，并对日志信息进行级别限制

* Appenders : 指定了日志将打印到控制台还是文件中

* Layout : 控制日志信息的显示格式

* 使用log4j步骤

```
* 下载，引入jar
* copy配置：log4j.properties
    log4j.rootLogger=INFO, A1
    log4j.appender.A1=org.apache.log4j.ConsoleAppender
    log4j.appender.A1.layout=org.apache.log4j.PatternLayout
    log4j.appender.A1.layout.ConversionPattern=%-4r %-5p [%t] %37c %3x - %m%n
```

```
 * 代码
    public static Logger logger=Logger.getLogger(Test1.class);// java.lang.Clas
    public static void main(String[] args) {
        // 日志级别
        // trace --> debug --> info --> warn -->error
        logger.trace("trace123");
        logger.debug("debug123");// 开发：调试信息，我们不希望在上线，被看到
        logger.info("info123");
        logger.warn("warn123");
        logger.error("error123");
    }
* 结果：
0    INFO  [main]                    com.lg.test1.Test1     - info123
1    WARN  [main]                    com.lg.test1.Test1     - warn123
1    ERROR [main]                    com.lg.test1.Test1     - error123
```

*Appender

```
* org.apache.log4j.ConsoleAppender：将日志信息输出到控制台。
* org.apache.log4j.FileAppender：将日志信息输出到一个文件。
* org.apache.log4j.DailyRollingFileAppender：将日志信息输出到一个日志文件，并且每天
* org.apache.log4j.RollingFileAppender：将日志信息输出到一个日志文件，并且指定文件的
* org.apache.log4j.WriteAppender：将日志信息以流格式发送到任意指定地方。

```

```
* RollingFileAppender
log4j.rootLogger=debug, A1
log4j.appender.A1=org.apache.log4j.RollingFileAppender
log4j.appender.A1.File=lg1.lg
log4j.appender.A1.Encoding=GBK
log4j.appender.A1.MaxFileSize=256KB
log4j.appender.A1.Append=true
log4j.appender.A1.MaxBackupIndex=20
log4j.appender.A1.layout=org.apache.log4j.PatternLayout
log4j.appender.A1.layout.ConversionPattern=%-4r %-5p [%t] %37c %3x - %m%n

```

```
while (true) {
            logger.trace("trace");
            logger.debug("debug");
            logger.info("info");
            logger.warn("warn");
            logger.error("error");
}

* DailyRollingFileAppender
log4j.rootLogger=debug, A1
log4j.appender.A1=org.apache.log4j.DailyRollingFileAppender
log4j.appender.A1.File=lg2.lg
log4j.appender.A1.DatePattern=yyyy-MM-dd'.log'
log4j.appender.A1.layout=org.apache.log4j.PatternLayout
log4j.appender.A1.layout.ConversionPattern=%-4r %-5p [%t] %37c %3x - %m%n
while (true) {
            logger.trace("trace");
            logger.debug("debug");
            logger.info("info");
            logger.warn("warn");
            logger.error("error");
        }
* 调整电脑的时间：观察

```

* Layout

  * HTMLLayout:格式化日志输出为HTML表格形式如下

```

log4j.rootLogger=warn, A1

log4j.appender.A1=org.apache.log4j.FileAppender
log4j.appender.A1.File=lg.html
log4j.appender.A1.layout=org.apache.log4j.HTMLLayout
log4j.appender.A1.layout.ConversionPattern=%-4r %-5p [%t] %37c %3x - %m%n
```

.og session start time Thu Aug 23 12:23:29 CST 2018

Time	Thread	Level	Category	Message
0	main	TRACE	com.hx.exception.Test2	trace
1	main	DEBUG	com.hx.exception.Test2	debug
1	main	INFO	com.hx.exception.Test2	info
1	main	WARN	com.hx.exception.Test2	warn
1	main	ERROR	com.hx.exception.Test2	error
1	main	FATAL	com.hx.exception.Test2	fatal

* SImpleLayout:以一种非常简单的方式格式化日志输出，它打印三项内容：级别-信息

```
log4j.rootLogger=warn, A1
log4j.appender.A1=org.apache.log4j.FileAppender
log4j.appender.A1.File=lg.log
log4j.appender.A1.layout=org.apache.log4j.SimpleLayout
log4j.appender.A1.layout.ConversionPattern=%-4r %-5p [%t] %37c %3x - %m%n
```

```
TRACE - trace
DEBUG - debug
INFO - info
WARN - warn
ERROR - error
FATAL - fatal
```

* PatternLayout : 根据指定的转换模式格式化日志输出，或者如果没有指定任何转换模式，就使用默认的转化模式格式。

```
Log4J采用类似C语言中的printf函数的打印格式格式化日志信息，打印参数如下：
# %m 输出代码中指定的消息
# %p 输出优先级，即DEBUG,INFO,WARN,ERROR,FATAL
# %r 输出自应用启动到输出该log信息耗费的毫秒数
# %c 输出所属的类目,通常就是所在类的全名
# %t 输出产生该日志事件的线程名
```

```
# %n 输出一个回车换行符，Windows平台为“\r\n”，Unix平台为“\n”
# %d 输出日志时间点的日期或时间，默认格式为ISO8601，也可以在其后指定格式
#    如：%d{yyyy年MM月dd日 HH:mm:ss,SSS}，输出类似：2012年01月05日 22:10:28,921
# %l 输出日志事件的发生位置，包括类目名、发生的线程，以及在代码中的行数
#    如：Testlog.main(TestLog.java:10)
# %F 输出日志消息产生时所在的文件名称
# %L 输出代码中的行号
# %x 输出和当前线程相关联的NDC(嵌套诊断环境),像java servlets多客户多线程的应用中
# %% 输出一个"%"字符
#
# 可以在%与模式字符之间加上修饰符来控制其最小宽度、最大宽度、和文本的对齐方式。如：
#  %5c: 输出category名称，最小宽度是5，category<5，默认的情况下右对齐
#  %-5c:输出category名称，最小宽度是5，category<5，"-"号指定左对齐,会有空格
#  %.5c:输出category名称，最大宽度是5，category>5，就会将左边多出的字符截掉，<5不会有
#  %20.30c:category名称<20补空格，并且右对齐，>30字符，就从左边交远销出的字符截掉
```

-

 * log4j2

  * Log4j2 的下载地址：https://logging.apache.org/log4j/2.x/download.html

```
* 依赖jar
   * log4j-api-2.11.1.jar
   * log4j-core-2.11.1.jar
* 测试
import org.apache.logging.log4j.LogManager;
import org.apache.logging.log4j.Logger;

public class Test1 {
      // 定义对 rootLogger 的静态引用
    private static final Logger logger = LogManager.getRootLogger();
    public static void main(String[] args) {
        logger.trace("trace Msg.");
        logger.debug("debug Msg.");
        logger.info("info Msg.");
        logger.warn("warn Msg.");
        logger.error("error Msg.");
    }
}
```

```

<?xml version="1.0" encoding="UTF-8"?>
<configuration status="WARN">
    <appenders>
        <Console name="console" target="SYSTEM_OUT">
            <PatternLayout
                pattern="%d{HH:mm:ss.SSS} [%t] %-5level %logger{36} - %msg%n" /
        </Console>
        <File name="log" fileName="log/test.log" append="false">
            <PatternLayout
                pattern="%d{HH:mm:ss.SSS} %-5level %class{36} %L %M - %msg%xEx%
        </File>
    </appenders>

    <loggers>
        <root level="info">
            <appender-ref ref="console" />
            <appender-ref ref="log" />
        </root>
    </loggers>
</configuration>

结果：
控制台
22:14:31.291 [main] INFO   - info Msg.
22:14:31.294 [main] WARN   - warn Msg.
22:14:31.294 [main] ERROR  - error Msg.
文件：
22:14:31.291 [main] INFO   - info Msg.
22:14:31.294 [main] WARN   - warn Msg.
22:14:31.294 [main] ERROR  - error Msg.
```

* RollingRandomAccessFile

* 按时间和文件大小滚动，效率比RollingFileAppender有很大的性能提升，官网宣称是20-200%

```

1

```

```
<Configuration status="WARN" monitorInterval="300">
    <properties>
        <property name="LOG_HOME">D:/logs</property>
        <property name="FILE_NAME">mylog</property>
    </properties>
    <Appenders>
        <Console name="Console" target="SYSTEM_OUT">
            <PatternLayout pattern="%d{HH:mm:ss.SSS} [%t] %-5level %logger{36}
        </Console>
        <RollingRandomAccessFile name="MyFile"
            fileName="${LOG_HOME}/${FILE_NAME}.log"
            filePattern="${LOG_HOME}/$${date:yyyy-MM}/${FILE_NAME}-%d{yyyy-MM-d
            <PatternLayout
                pattern="%d{yyyy-MM-dd HH:mm:ss.SSS} [%t] %-5level %logger{36}
            <Policies>
                <TimeBasedTriggeringPolicy interval="1" />
                <SizeBasedTriggeringPolicy size="10 MB" />
            </Policies>
            <DefaultRolloverStrategy max="20" />
        </RollingRandomAccessFile>
    </Appenders>

    <Loggers>
        <Logger name="mylog" level="trace" additivity="false">
            <AppenderRef ref="MyFile" />
        </Logger>
        <Root level="error">
            <AppenderRef ref="Console" />
        </Root>
    </Loggers>
</Configuration>

Logger logger = LogManager.getLogger("mylog");
        for (;;) {
            logger.trace("trace level");
            logger.debug("debug level");
            logger.info("info level");
            logger.warn("warn level");
            logger.error("error level");
            logger.fatal("fatal level");
```

```
}
<properties>定义了两个常量方便后面复用
RollingRandomAccessFile的属性：
fileName  指定当前日志文件的位置和文件名称
filePattern  指定当发生Rolling时，文件的转移和重命名规则
SizeBasedTriggeringPolicy  指定当文件体积大于size指定的值时，触发Rolling
DefaultRolloverStrategy  指定最多保存的文件个数
TimeBasedTriggeringPolicy  这个配置需要和filePattern结合使用，
注意filePattern中配置的文件重命名规则是${FILE_NAME}-%d{yyyy-MM-dd HH-mm}-%i，
最小的时间粒度是mm，即分钟，TimeBasedTriggeringPolicy指定的size是1，
结合起来就是每1分钟生成一个新文件。如果改成%d{yyyy-MM-dd HH}，最小粒度为小时，
则每一个小时生成一个文件。
```

* 能够理解集合的概述

  * 概述：集合是用来存储引用类型数据的容器。

  * 集合分类

    * Collection集合（单列集合）：存储数据时,是单个存储的

    * Map集合（双列集合），存储数据时是按<键,值>对的形式存储的, <”xiaohei”, 18>

* 能够掌握集合和数组的区别

    * 数组的长度是固定的，集合的长度是可变的。

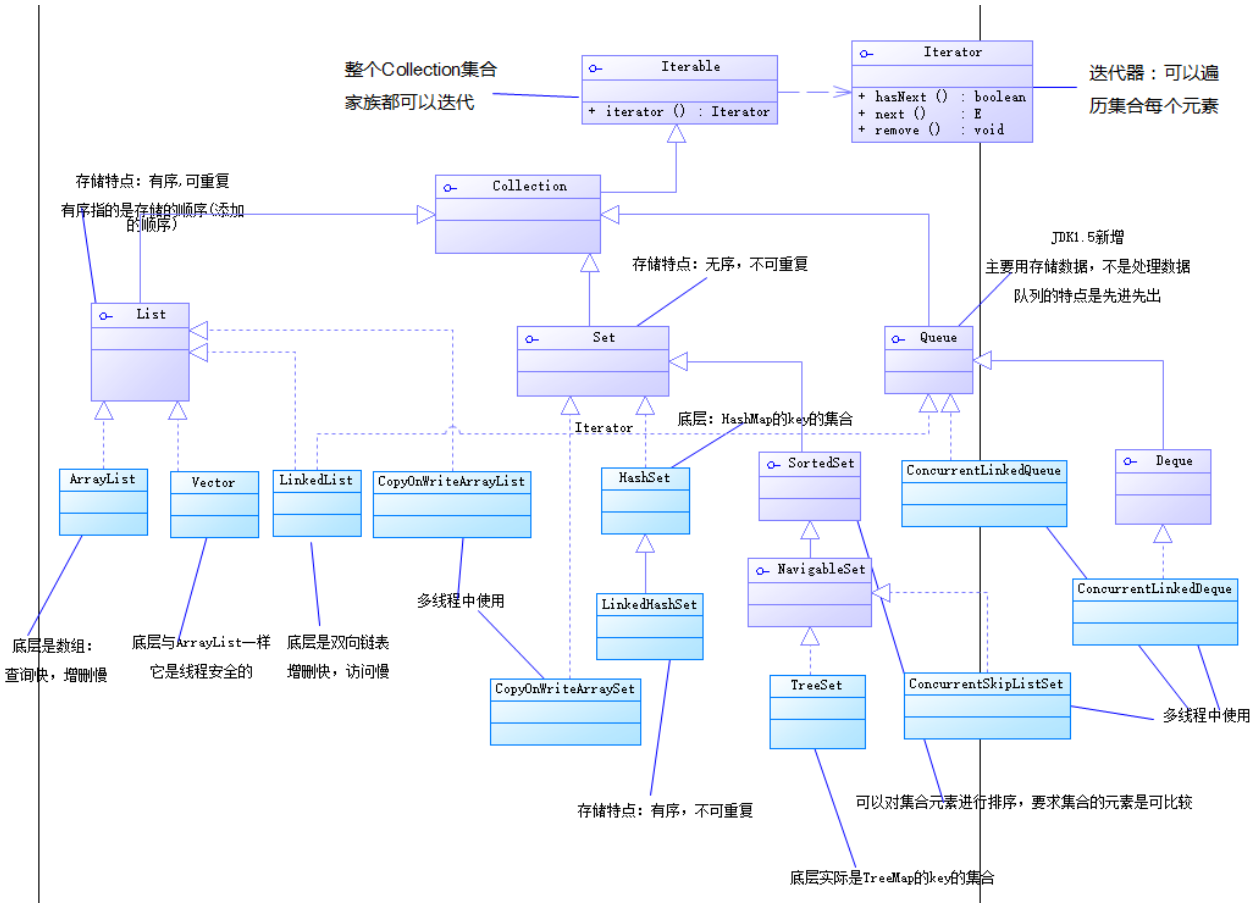
    * 很多说数组只能存储相同数据类型的数据，这句话话对不对？

```
public static void main(String[] args) {
        // 数组也是可以存储不同类型的数据的
        Object objs[]=new Object[5];
        objs[0]="123";
        objs[1]=1;
        System.out.println(objs[0]);
        System.out.println(objs[1]);
    }

结果：
123
1
```

* 能够了解集合框架

  * Collection 集合框架图



    * Collection

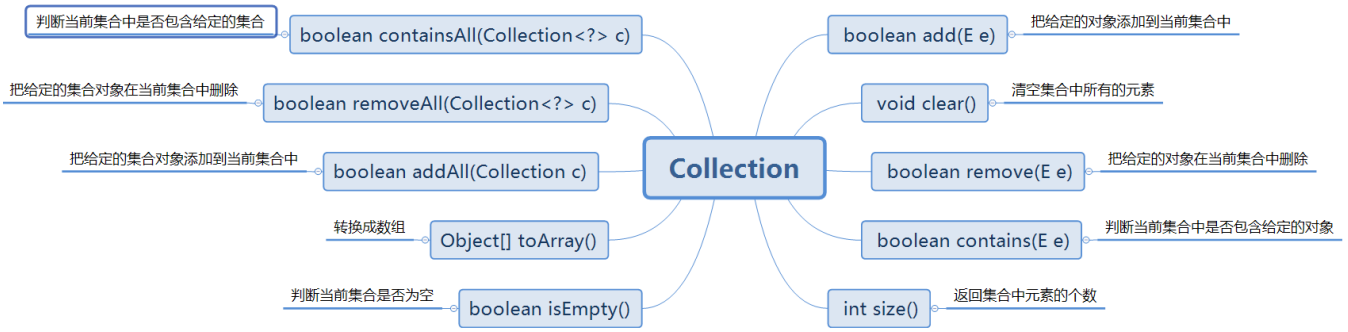
      * List：ArrayList，LinkedList

      * Set ：HashSet，LinkedHashSet

    * Map

* 能够掌握Collection集合常用的方法

    * add，remove，contains，size，clear，isEmpty,addAll,removeAll,containsAll,toArray



* 案例演示

```
public static void main(String[] args) {
        //add，remove，contains，size，clear，isEmpty,addAll,removeAll,containsA
        Collection<String> container=new ArrayList<String>();
        // add:东邪西毒南帝北丐中神通
        container.add("东邪");
        container.add("西毒");
        container.add("南帝");
        container.add("北丐");
        container.add("中神通");
        System.out.println(container);
        // remove
        container.remove("西毒");
        System.out.println(container);
        // contains
        System.out.println(container.contains("南帝"));
        // size
        System.out.println(container.size());
        // clear
        container.clear();
        // isEmpty
        System.out.println(container.isEmpty());

        //addAll
        container.add("东邪");
```

```
        container.add("西毒");
        container.add("南帝");
        container.add("北丐");
        container.add("中神通");
        Collection<String> container2=new ArrayList<String>();
        container2.add("A");
        container2.add("B");
        container2.add("C");
        container2.add("D");
        container2.add("E");
        container.addAll(container2);
        System.out.println(container);
        // containsAll,removeAll
        Collection<String> container3=new ArrayList<String>();
        container3.add("A");
        container3.add("B");
        container3.add("C");
        System.out.println(container.containsAll(container3));
        container.removeAll(container3);
        System.out.println(container);

        //toArray
        Object[] objs = container3.toArray();
        System.out.println(objs[1]);
        for(Object obj:objs) {
            System.out.println(obj);
        }
    }

结果:
[东邪, 西毒, 南帝, 北丐, 中神通]
[东邪, 南帝, 北丐, 中神通]
true
4
true
[东邪, 西毒, 南帝, 北丐, 中神通, A, B, C, D, E]
true
[东邪, 西毒, 南帝, 北丐, 中神通, D, E]
B
A
```

```
B
C

```

* 能够使用迭代器Iterator和增强型for对集合进行遍历

  * 迭代概述：

```
迭代：Collection集合元素的通用获取方式。
在取元素之前先要判断集合中有没有元素，如果有，就把这个元素取出来，继续在判断，
如果还有就再取出出来。一直把集合中的所有元素全部取出。这种取出方式专业术语称为迭代
```

  * 画图解释迭代器原理

    * 在此图的基础上画


```
public static void main(String[] args) {
        Collection<String> container=new ArrayList<String>();
        // add:东邪西毒南帝北丐中神通
        container.add("东邪");
```

```
        container.add("西毒");
        container.add("南帝");
        container.add("北丐");
        container.add("中神通");

        Iterator<String> ite = container.iterator();
        while(ite.hasNext()) {
            String value=ite.next();
            System.out.println(value);
        }
//      ite.next();// java.util.NoSuchElementException
        System.out.println("--------------------------------");
        for(String v:container) {
            System.out.println(v);
        }
    }

结果:
东邪
西毒
南帝
北丐
中神通
--------------------------------
东邪
西毒
南帝
北丐
中神通
```

*　增强for循环

* 增强for循环(也称for each循环)是JDK1.5以后出来的一个高级for循环，专门用来遍历数组和集合的。它的内部原理其实是个Iterator迭代器，所以在遍历的过程中，不能对集合中的元素进行增删操作。

```
Collection<String> container=new ArrayList<String>();
        // add:东邪西毒南帝北丐中神通
        container.add("东邪");
```

```
        container.add("西毒");
        container.add("南帝");
        container.add("北丐");
        container.add("中神通");
for(String v:container) {
            container.remove("北丐");
            System.out.println(v);
}
结果并报异常：
东邪
Exception in thread "main" java.util.ConcurrentModificationException
    at java.util.ArrayList$Itr.checkForComodification(Unknown Source)
    at java.util.ArrayList$Itr.next(Unknown Source)
    at com.lg.test.Test3.main(Test3.java:24)

```