dontpanic study notes

Java

Java 和 JDK 的关系

JDK（Java Development Kit）Java 开发工具包，它包括：编译器、Java 运行环境（JRE，Java Runtime Environment）、JVM（Java 虚拟机）监控和诊断工具等，而 Java 则表示一种开发语言。

Java 程序是怎么执行的?

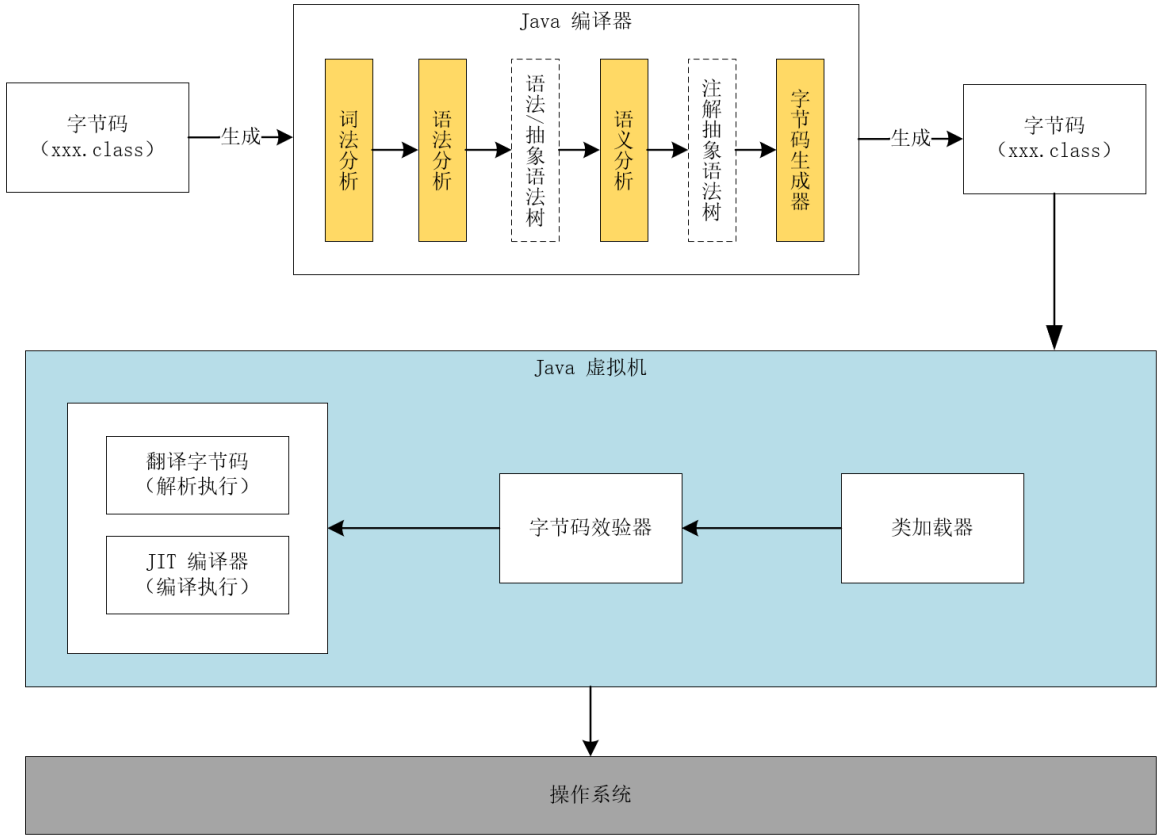
我们日常的工作中都使用开发工具（IntelliJ IDEA 或 Eclipse 等）可以很方便的调试程序，或者是通过打包工具把项目打包成 jar 包或者 war 包，放入 Tomcat 等 Web 容器中就可以正常运行了，但你有没有想过 Java 程序内部是如何执行的?

其实不论是在开发工具中运行还是在 Tomcat 中运行，Java 程序的执行流程基本都是相同的，它的执行流程如下：

先把 Java 代码编译成字节码，也就是把 .java 类型的文件编译成 .class 类型的文件。这个过程的大致执行流程：Java 源代码 → 词法分析器 → 语法分析器 → 语义分析器 → 字符码生成器 → 最终生成字节码，其中任何一个节点执行失败就会造成编译失败；
把 class 文件放置到 Java 虚拟机，这个虚拟机通常指的是 Oracle 官方自带的 Hotspot JVM；
Java 虚拟机使用类加载器（Class Loader）装载 class 文件；
类加载完成之后，会进行字节码效验，字节码效验通过之后 JVM 解释器会把字节码翻译成机器码交由操作系统执行。但不是所有代码都是解释执行的，JVM 对此做了优化，比如，以 Hotspot 虚拟机来说，它本身提供了 JIT（Just In Time）也就是我们通常所说的动态编译器，它能够在运行时将热点代码编译为机器码，这个时候字节码就变成了编译执行。

Java 执行流程：Java 源代码（.java）→ 编译 → Java 字节码（.class） →通过 JVM（Java 虚拟机）运行 Java 程序。每种类型的服务器都会运行一个 JVM，Java 程序只需要生成 JVM 可以执行的代码即可，JVM 底层屏蔽了不同服务器类型之间的差异，从而可以在不同类型的服务器上运行一套 Java 程序。

Java 程序执行流程图如下：



Client

WebSocket

用于跟 web 端通信传文件，在 conclusion.md 的 websocket 部分有说明

META-INF

好像是 java 方面的东西，不太用管

JAR 文件就是 Java Archive File，顾名思意，它的应用是与 Java 息息相关的，是 Java 的一种文档格式。JAR 文件非常类似 ZIP 文件——准确的说，它就是 ZIP 文件，所以叫它文件包。JAR 文件与 ZIP 文件唯一的区别就是在 JAR 文件的内容中，包含了一个 META-INF/MANIFEST.MF 文件，这个文件是在生成 JAR 文件的时候自动创建的.

创建可执行 JAR

创建一个可执行 JAR 很容易。首先将所有应用程序代码放到一个目录中。假设应用程序中的主类是 com.mycompany.myapp.Sample 。您要创建一个包含应用程序代码的 JAR 文件并标识出主类。为此，在某个位置(不是在应用程序目录中)创建一个名为 manifest 的文件，并在其中加入以下一行:

> Main-Class: com.mycompany.myapp.Sample 然后，像这样创建 JAR 文件：
>
> jar cmf manifest ExecutableJar.jar application-dir
>
> 所要做的就是这些了，现在可以用java -jar执行这个JAR文件ExecutableJar.jar。
>
> 一个可执行的 JAR 必须通过 menifest 文件的头引用它所需要的所有其他从属 JAR。如果使用了 -jar 选项，那么环境变量 CLASSPATH 和在命令行中指定的所有类路径都被 JVM 所忽略。

fileDetector

文件夹监控包，跟 DFS OSH2017 那边应该没区别，在那边的答辩报告里有说明

没用到

connect

有些区别，在那边的答辩报告里也有点说明

com

文件解、编码，这部分工作在 dontpanic 是在 web 端利用 js 或 webassembly 实现的，在 clinet 中的 com 的用处还不明确

没用到

client

也有点区别但区别不大

web/WebContent

调研 DFS

首先调研了一下 DFS 那边的 web 端实现：

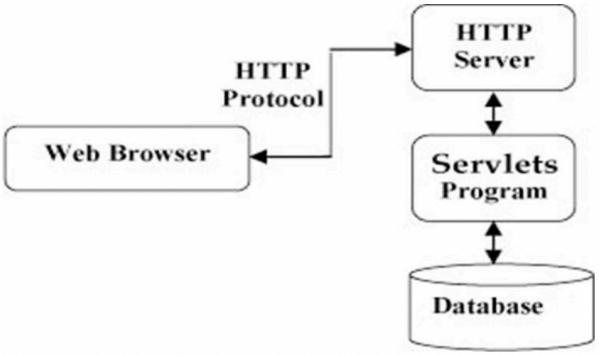
> 用户通过网页端访问
>
> web 采用 tomcat，结合 stuct 架构，采用 bootstrap 主题（应该是 struts 吧）
>
> 客户端将 icon.xpm 放到共享文件夹
>
> 网页显示注册登录界面

登录成功后进入主界面，显示虚拟分享文件目录

`......`(详见 DFS 的 report 中的使用说明)(这个例子应该就是先把 xpm 文件放到共享文件夹使得它被上传到 client 再下载下来吧)

WEB服务

◘ Tomcat WEB应用服务器

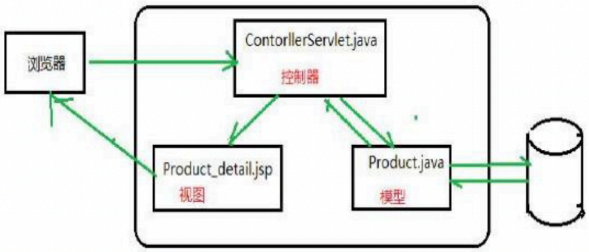


- 本身支持一般的http服务
- 提供JSP的JAVA运行环境和JavaBean的运行环境
- 响应浏览器请求，调用服务器端java函数
- 在以上基础上，搭建动态网站

重点：tomcat 可以调用服务器端的 java 函数!

WEB服务

◘ Struts2 动态网站网站应用调度框架



- Tomcat基础上
- 采用MVC设计模式
- 控制器从视图读取数据，控制用户输入，向模型发送数据
- 管理复杂的应用程序，可以在一个时间内专门关注一个方面
- 简化了分组开发，不同开发人员可同时开发视图、控制器逻辑和业务逻辑

WEB服务

◘ BOOTSTRAP网页主题

- 基于 HTML、CSS、JAVASCRIPT
- 适用于快速开发+可视化开发+提供可重用组件
- 响应式 CSS 能够自适应于不同尺寸电脑和手机
- 所有的主流浏览器都支持 Bootstrap

WEB服务

◘ JQery+AJAX异步C/S通讯+JSON+MySql

- JQery-----"写的少，做的多"，极大地简化了 JavaScript 编程
- AJAX-----不重载全部页面，实现对部分网页的更新
- JSON-----轻量级的数据交换格式
- MySql-----关系数据库管理系统

maven 目录结构

约定配置

Maven 提倡使用一个共同的标准目录结构，Maven 使用约定优于配置的原则，大家尽可能的遵守这样的目录结构。如下所示：

目录	目的
${basedir}	存放pom.xml和所有的子目录
${basedir}/src/main/java	项目的java源代码
${basedir}/src/main/resources	项目的资源，比如说property文件，springmvc.xml
${basedir}/src/test/java	项目的测试类，比如说Junit代码
${basedir}/src/test/resources	测试用的资源
${basedir}/src/main/webapp/WEB-INF	web应用文件目录，web项目的信息，比如存放web.xml、本地图片、jsp视图页面
${basedir}/target	打包输出目录
${basedir}/target/classes	编译输出目录
${basedir}/target/test-classes	测试编译输出目录
Test.java	Maven只会自动运行符合该命名规则的测试类
~/.m2/repository	Maven默认的本地仓库目录位置

跟本目录的结构一样

pom.xml

maven 的东西，***暂时不知道 mysql 版本要不要改***

iml 文件

不用管，应该是自己生成的东西

mysql.sql

用处暂时未知，看了一下 web 下跟 mysql 有关的除了 xml 和 iml 似乎就只有 main 下的 java/database/Query.java 里的 mysql

target

应该是生成的 class 文件，不用看

src/main

重点！！！

- resources

  resources 中存了一个 struts.xml，应该是说明了前端 webapp 中的 action 对应了 java 中的哪个类吧，所以 java 目录下没有 main class 也能理解是为什么了，毕竟 java 是被调用者，不需要指定它的主入口

- java 和 webapp

  两部分结合一起看，对照着最后放到 tomcat 的 web-app-name-2020 看

  index.html 通过 action 跳到 java 下的 userManagement.UserReg，没有找到能跳到 userManagement.UserLogin 的 action，但在 index_ajax.js 有如下代码片段：

```
$.ajax({
url:"UserLogin.action",
type:"POST",
data:form,
dataType:"text",
processData:false,
contentType:false,
success:function(databack){
  var obj = $.parseJSON(databack);
  var feedback = obj.result;
  if(feedback==str)
    window.location.href='jsp/majorPage.jsp';
  //格式是ｊｓｏｎ　输出反馈信息到ｃｏｎｓｏｌｅ
  else
```

```
    $("#statusFeedback").text(feedback);
}
});
```

应该就是通过这个 url 调用的

index.html 对应的 js 是 js/index_ajax.js（当然还有一些跟 jquery 有关的 js，在 html 开头都可以看到，这部分当黑盒就行），可以看到用户注册后页面应该不会跳转，只有登陆后页面才会跳转到 jsp/majorPage.jsp，这个就是主页面

> JSP和HTML之间的主要区别在于**JSP是一种创建动态Web应用程序的技术，而HTML是用于创建Web页面结构的标准标记语言。** 简而言之，JSP文件是一个带有Java代码的HTML文件。
>
> 参考: www.php.cn/website-design-ask-414886.html

majorPage 中 button 主要在 172 - 177 行，button 映射到 js/majorPage_ajax.js 中 418 行开始的 ready 函数，对应着不同的操作，还有一部分映射比如 curr_path, file_list_body 等就是 "子页面" 的 button 吧

majorPage_ajax 中调用 java 程序的代码都有固定的格式，以 FileUploader 为例（对应到 userManagement.FileDownloader）：

```
$.ajax({
  url: "FileUploader!uploadRegister.action", // ！应该是指定调用类的 uploadRegister 方法
  type: "POST",
  data: uploadForm,
  dataType: "text",
  processData: false,
  contentType: false,
  async: false,                //此处采用同步查询进度
  success: function (databack) {
    var retFileInfo = $.parseJSON(databack);
    let result = retFileInfo.result;
    deviceArray = retFileInfo.devices.forms;
    fileId=retFileInfo.fileId;
    console.log(result);
    //alert(result);
  }
});
```

到这里 web 前端的逻辑就差不多完了，调用 java 的部分可以有需要再看，毕竟那部分我们不怎么需要改

这里以 download file 为例说明一下逻辑：（不保证完全正确）

首先用户登陆后在网页端 click download button，由 majorPage_ajax.jsp 跳到 majorPage_ajax.js 执行相应的程序

首先是跳到 418 行的 ready 函数内执行：

```
$("#button_download").click(function(){
    fileDownload();
});
```

在 fileDownload() 函数内调用 java 中的 userManagement.FileDownloader，作用应该是跟服务器通信，获取存储节点的编号等等的，这里还没有跟存储节点通信，也即还没有下载文件：

```
$.ajax({
  url:"FileDownloader!downloadRegister.action",
  type:"POST",
  data:form,
  dataType:"text",
  processData:false,
  contentType:false,
  async: false,                //此处采用同步查询进度
  success:function(databack){
    fileInfo = $.parseJSON(databack);
    //alert(result);
  }
});
```

获取了这些数据后就开始跟存储节点通信，从多个存储节点中下载文件：

```
for(var i=0;i<deviceArray.length;i++)
{
  console.log(deviceArray[i]);
  let
received_bytes=WebSocketDownload(deviceArray[i].ip,deviceArray[i].port,d
eviceArray[i].filename,content,digest,deviceArray[i].fragmentId);
  //console.log(received_bytes);
  console.log('Back');
  //console.log(content[deviceArray[i].fragmentId];
  //createAndDownloadFile(deviceArray[i].filename, 'jpg',
received_bytes)
}
```

WebSocketDownload 是在 majorPage.js 中定义的函数，应该就是用来下载文件的

下载完之后 decode：

```
let downloadTimeoutId =setTimeout(function(){

decodeFile(fileInfo.name,fileInfo.fileType,fileInfo.nod,fileInfo.noa,con
tent,digest,fileInfo.fileSize);
}, 10000)
```

decodeFile() 定义如下：

```
function decodeFile(fileName, fileType, numOfDivision, numOfAppend,
content, digest, fileSize) {
  //clean wrong parts
  var errors = 0;
  /*
  for (var i = 0; i < content.length; i++) {
    if (digest[i] != objectHash.MD5(content[i])) {
      errors += 1;
      content[i] = new Uint8Array(content[i].length);
    }
  }*/

  //console.log(content);
  const t5 = Date.now();//Decode timing start

  var contentView=new Array(content.length);
  for(var i=0;i<content.length;i++){
    contentView[i]=new Uint8Array(content[i]);
  }
  //var decoded = erasure.recombine(contentView, fileSize,
numOfDivision, numOfAppend);
  var decoded = callDecoder(contentView, numOfDivision, numOfAppend);
  //console.log(decoded);
  if (decoded.length > fileSize)
    decoded = decoded.subarray(0, fileSize);
  const t6 = Date.now();//Decode timing end

  // after decoded, download the file and show info(time, errors)
  createAndDownloadFile(fileName, fileType, decoded);

  if (document.getElementById("decode") != null)
    document.getElementById("decode").innerHTML += "Decode with " +
errors + " errors succeeded in " + (t6 - t5) + "mS</br>";
  console.log("Erasure decode took " + (t6 - t5) + " mS");
  return Promise.resolve(true);
}
```

这里 decode 主要调用了 callDecoder 这个函数，但这个函数在文件夹中找不到定义，怀疑是没给出源文件直接生成了可执行文件 mycoder.wasm，所以可以直接调用但找不到定义在哪，建议找学长要一下

然后是 createAndDownloadFile，应该就是 websocket 把 decode 前的文件下载到了不知道哪里，再调用这个函数把 decode 过后的文件下载到用户指定的文件路径吧，然后 download 就完成了