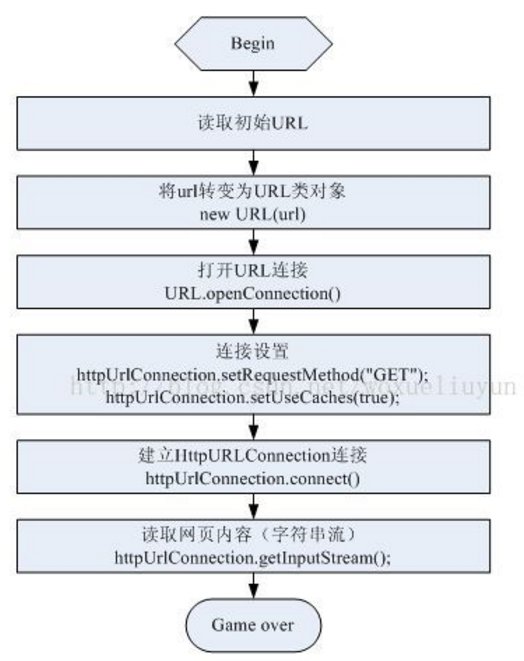
## 请求响应流程



## 设置连接参数的方法

* setAllowUserInteraction
* setDoInput
* setDoOutput
* setIfModifiedSince
* setUseCaches
* setDefaultAllowUserInteraction
* setDefaultUseCaches

## 设置请求头或响应头

HTTP请求允许一个key带多个用逗号分开的values，但是HttpURLConnection只提供了单个操作的方法：

* setRequestProperty(key,value)
* addRequestProperty(key,value)

setRequestProperty和addRequestProperty的区别就是，setRequestProperty会覆盖已经存在的key的所有values，有清零重新赋值的作用。而addRequestProperty则是在原来key的基础上继续添加其他value。

## 发送URL请求

建立实际连接之后，就是发送请求，把请求参数传到服务器，这就需要使用outputStream把请求参数传给服务器：

* getOutputStream

## 获取响应

请求发送成功之后，即可获取响应的状态码，如果成功既可以读取响应中的数据，获取这些数据的方法包括：

* getContent
* getHeaderField
* getInputStream

对于大部分请求来说，getInputStream和getContent是用的最多的。

相应的信息头用以下方法获取：

* getContentEncoding
* getContentLength
* getContentType
* getDate
* getExpiration
* getLastModifed

## HttpURLConnection

任何网络连接都需要经过socket才能连接，HttpURLConnection不需要设置socket，所以，HttpURLConnection并不是底层的连接，而是在底层连接上的**一个请求**。这就是为什么HttpURLConneciton只是一个抽象类，自身不能被实例化的原因。HttpURLConnection只能通过URL.openConnection()方法创建具体的实例。

虽然**底层**的网络连接可以被多个HttpURLConnection实例共享，但每一个HttpURLConnection实例只能发送一个请求。请求结束之后，应该调用HttpURLConnection实例的InputStream或OutputStream的close()方法以释放请求的网络资源，不过这种方式对于持久化连接没用。对于持久化连接，得用disconnect()方法关闭底层连接的socket。

### 创建HttpURLConnection

1. URL url = **new** URL("http://localhost:8080/xxx.do");
2. URLConnection rulConnection = url.openConnection();// 此处的urlConnection对象实际上是根据URL的
3. // 请求协议(此处是http)生成的URLConnection类
4. // 的子类HttpURLConnection,故此处最好将其转化
5. // 为HttpURLConnection类型的对象,以便用到
6. // HttpURLConnection更多的API.如下:
7. HttpURLConnection httpUrlConnection = (HttpURLConnection) rulConnection;

### 设置HttpURLConnection参数

1. // 设定请求的方法为"POST"，默认是GET
2. httpUrlConnection.setRequestMethod("POST");
3. // 设置是否向httpUrlConnection输出，因为这个是post请求，参数要放在
4. // http正文内，因此需要设为true, 默认情况下是false;
5. httpUrlConnection.setDoOutput(**true**);
6. // 设置是否从httpUrlConnection读入，默认情况下是true;
7. httpUrlConnection.setDoInput(**true**);
8. // Post 请求不能使用缓存
9. httpUrlConnection.setUseCaches(**false**);
10. // 设定传送的内容类型是可序列化的java对象
11. // (如果不设此项,在传送序列化对象时,当WEB服务默认的不是这种类型时可能抛java.io.EOFException)
12. httpUrlConnection.setRequestProperty("Content-type", "application/x-java-serialized-object");
13. // 连接，从上述url.openConnection()至此的配置必须要在connect之前完成，
14. httpUrlConnection.connect();

### URLConnection建立连接

1. // 此处getOutputStream会隐含的进行connect(如同调用上面的connect()方法，
2. // 所以在开发中不调用上述的connect()也可以)。
3. OutputStream outStrm = httpUrlConnection.getOutputStream();

### HttpURLConnection发送请求

1. // 现在通过输出流对象构建对象输出流对象，以实现输出可序列化的对象。
2. ObjectOutputStream objOutputStrm = **new** ObjectOutputStream(outStrm);
3. // 向对象输出流写出数据，这些数据将存到内存缓冲区中
4. objOutputStrm.writeObject(**new** String("我是测试数据"));
5. // 刷新对象输出流，将任何字节都写入潜在的流中（些处为ObjectOutputStream）
6. objOutputStm.flush();
7. // 关闭流对象。此时，不能再向对象输出流写入任何数据，先前写入的数据存在于内存缓冲区中,
8. // 在调用下边的getInputStream()函数时才把准备好的http请求正式发送到服务器
9. objOutputStm.close();

### HttpURLConneciton获取响应

**int** resultCode = httpUrlConnection.getResponseCode();

JSONObject result = **new** JSONObject();

**if** (HttpURLConnection.***HTTP\_OK*** == resultCode) {

String readLine = **new** String();

StringBuffer readbuf = **new** StringBuffer();

BufferedReader responseReader = **new** BufferedReader(

**new** InputStreamReader(httpUrlConnection.getInputStream(), "UTF-8"));

**while** ((readLine = responseReader.readLine()) != **null**) {

readbuf.append(readLine);

}

result = **new** JSONObject(readbuf.toString());

**if**(result.getInt("code")==0) {

System.***out***.println("操作成功"+result.getString("msg"));

}**else** **if**((Integer)result.get("code")==1) {

System.***out***.println("操作失败"+result.getString("msg"));

}

responseReader.close();

}**else** {

System.***out***.println("请求失败"+resultCode+""+httpUrlConnection.getResponseMessage());

}

httpUrlConnection.disconnect();

## 总结:

a: HttpURLConnection的connect()函数，实际上只是建立了一个与服务器的tcp连接，并没有实际发送http请求。无论是post还是get，http请求实际上直到HttpURLConnection的getInputStream()这个函数里面才正式发送出去。

b:在用POST方式发送URL请求时，URL请求参数的设定顺序是重中之重，对connection对象的一切配置（那一堆set函数）    都必须要在connect()函数执行之前完成。而对outputStream的写操作，又必须要在inputStream的读操作之前。    这些顺序实际上是由http请求的格式决定的。    如果inputStream读操作在outputStream的写操作之前，会抛出例外：    java.net.ProtocolException: Cannot write output after reading input.......

c:http请求实际上由两部分组成，  
    一个是http头，所有关于此次http请求的配置都在http头里面定义,一个是正文content。    connect()函数会根据HttpURLConnection对象的配置值生成http头部信息，因此在调用connect函数之前，    就必须把所有的配置准备好。

d: 在http头后面紧跟着的是http请求的正文，正文的内容是通过outputStream流写入的，    实际上outputStream不是一个网络流，充其量是个字符串流，往里面写入的东西不会立即发送到网络，    而是存在于内存缓冲区中，待outputStream流关闭时，根据输入的内容生成http正文。至此，http请求的东西已经全部准备就绪。在getInputStream()函数调用的时候，就会把准备好的http请求正式发送到服务器了，然后返回一个输入流，用于读取服务器对于此次http请求的返回信息。由于http请求在getInputStream的时候已经发送出去了（包括http头和正文），因此在getInputStream()函数之后对connection对象进行设置（对http头的信息进行修改）或者写入outputStream（对正文进行修改）都是没有意义的了，执行这些操作会导致异常的发生。