我们来看具体获取Response的getResponseWithInterceptorChain方法

```
 Response getResponseWithInterceptorChain() throws IOException {
    // Build a full stack of interceptors.
    List<Interceptor> interceptors = new ArrayList<>();
    interceptors.addAll(client.interceptors());//1
    interceptors.add(retryAndFollowUpInterceptor);//2
    interceptors.add(new BridgeInterceptor(client.cookieJar()));//3
    interceptors.add(new CacheInterceptor(client.internalCache()));//4
    interceptors.add(new ConnectInterceptor(client));//5
    if (!forWebSocket) {
      interceptors.addAll(client.networkInterceptors());
    }
    interceptors.add(new CallServerInterceptor(forWebSocket));//6
    Interceptor.Chain chain = new RealInterceptorChain(interceptors, null, null,
null, 0,
        originalRequest, this, eventListener, client.connectTimeoutMillis(),
        client.readTimeoutMillis(), client.writeTimeoutMillis());//7
    return chain.proceed(originalRequest);
  }
```

point one 添加**okhttpclint**中我们自定义的**interceptor**。

point two 添加负责失败和重新定向的**RetryAndFollowUpInterceptor**

point three 添加负责把用户构造的请求转换为发送到服务器的请求、把服务器返回的响应转换为用户友好的响应的**BridgeInterceptor**

ponit four 添加**collection**负责读取缓存直接返回、更新缓存的**CacheInterceptor**

ponit five 添加负责和服务器建立连接的**ConnectInterceptor**；

point six 添加负责向服务器发送请求数据、读取响应数据的CallServerInterceptor。

point seven 构造了处理拦截器的链子RealInterceptorChain。

前几步都是向list内添加Interceptor。最后用这个list，加上client的部分参数。new了一个RealInterceptorChain。并通过chain.proceed(originalRequest)获取到了response。

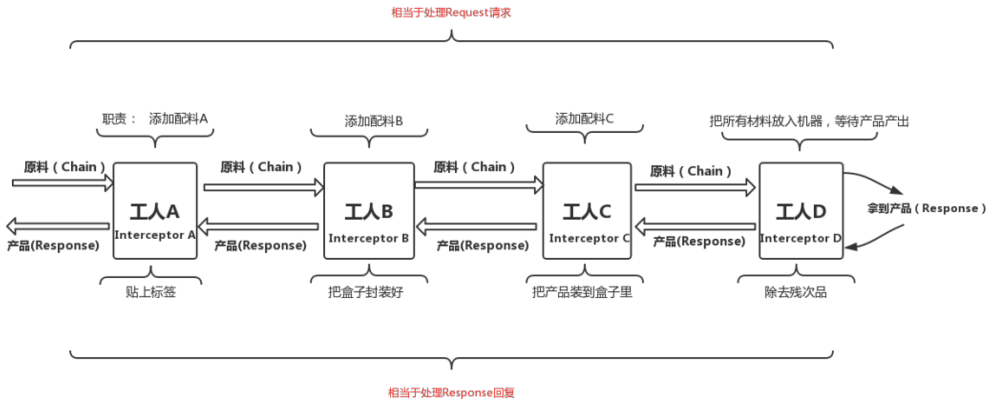
同样，我们来看RealInterceptorChain所实现的接口Chain：

```
public interface Interceptor {
  Response intercept(Chain chain) throws IOException;
  interface Chain {
    Request request();
    Response proceed(Request request) throws IOException;
//忽略代码
.....
}
```

Chain接口是作为内部接口放在Interceptor接口内的。Interceptor负责拦截加工Chain「request和response」。在RealInterceptorChain调用proceed方法创建下一个节点的Chain，然后再下一个Chain中，定位到下一个interceptor,在下一个interceptor的intercept方法获取中response。一环扣一环，直到最后一个interceptor即CallServerInterceptor进行网络请求获取response。

```
  RealInterceptorChain next = new RealInterceptorChain(interceptors,
streamAllocation, httpCodec,
        connection, index + 1, request, call, eventListener, connectTimeout,
readTimeout,
        writeTimeout);
```

```
    Interceptor interceptor = interceptors.get(index);
    Response response = interceptor.intercept(next);
//责任链模式
```



我们具体来看Chain的实现类，RealInterceptorChain

作者给他的描述为/**
 * A concrete interceptor chain that carries the entire interceptor chain: all application
 * interceptors, the OkHttp core, all network interceptors, and finally the network caller.
 */

持有变量和构造方法

```
 private final List<Interceptor> interceptors;
 private final StreamAllocation streamAllocation;
 private final HttpCodec httpCodec;
 private final RealConnection connection;
 private final int index;
```

```
 private final Request request;
 private final Call call;
 private final EventListener eventListener;
 private final int connectTimeout;
 private final int readTimeout;
 private final int writeTimeout;
 private int calls;

  public RealInterceptorChain(List<Interceptor> interceptors, StreamAllocation
streamAllocation,
      HttpCodec httpCodec, RealConnection connection, int index, Request request, Call
call,
      EventListener eventListener, int connectTimeout, int readTimeout, int writeTimeout) {
  }
```

通过分析代码，我们看到
StreamAllocation最开始为空，会在RetryAndFollowUpInterceptor中new出来
httpCodec最开始也为空，会在ConnectInterceptor中初始化
RealConnection，同样是在RealConnection中初始化的。