Manipulating Java DNS Cache

Usage:

```
try {
        String[] ips = {"192.168.1.10"};
        String hostName = "myhost.com";
        InetAddressCacheUtil.setInetAddressCache(hostName, ips, InetAddressCacheUtil.NEVER_EXPIRATION);
}catch (Exception ex){
        ...
}
```

Code:

```
package io.kafka.connect.log.anlaytics.util;

import java.lang.reflect.Constructor;
import java.lang.reflect.Field;
import java.lang.reflect.InvocationTargetException;
import java.net.InetAddress;
import java.net.UnknownHostException;
import java.util.concurrent.ConcurrentMap;
import java.util.concurrent.ConcurrentSkipListSet;

public class InetAddressCacheUtil {

    public static final long NEVER_EXPIRATION = Long.MAX_VALUE;

    public static void setInetAddressCache(String host, String[] ips, long expireMillis) throws Exception {

        Object cachedAddresses = newCachedAddresses(host, ips, expireMillis);
        getCacheOfInetAddress().put(host, cachedAddresses);
        getInetAddressExpiries().add(cachedAddresses);
    }

    private static Object newCachedAddresses(String host, String[] ips, long expiration)
            throws ClassNotFoundException, UnknownHostException, IllegalAccessException,
            InvocationTargetException, InstantiationException {

           final Class<?> clazz = Class.forName("java.net.InetAddress$CachedAddresses");
           final Constructor<?> constructor = clazz.getDeclaredConstructors()[0];
           constructor.setAccessible(true);
           return constructor.newInstance(host, toInetAddressArray(host, ips), expiration);

    }

    private static ConcurrentMap<String, Object> getCacheOfInetAddress()
            throws NoSuchFieldException, IllegalAccessException {
        return (ConcurrentMap<String, Object>) getAddressCacheAndExpirySet()[0];
    }

    private static Object[] getAddressCacheAndExpirySet() throws NoSuchFieldException, IllegalAccessException {
            Object[] addressCacheAndExpirySet = null;

            final Field cacheField = InetAddress.class.getDeclaredField("cache");
            cacheField.setAccessible(true);

            final Field expirySetField = InetAddress.class.getDeclaredField("expirySet");
            expirySetField.setAccessible(true);

            addressCacheAndExpirySet = new Object[]{
                    cacheField.get(InetAddress.class),
                    expirySetField.get(InetAddress.class)
            };

            return addressCacheAndExpirySet;
    }
```

```
    private static ConcurrentSkipListSet<Object> getInetAddressExpiries()
            throws NoSuchFieldException, IllegalAccessException {
        return (ConcurrentSkipListSet<Object>) getAddressCacheAndExpirySet()[1];
    }

    static InetAddress[] toInetAddressArray(String host, String[] ips) throws UnknownHostException {
        InetAddress[] addresses = new InetAddress[ips.length];
        for (int i = 0; i < addresses.length; i++) {

            addresses[i] = InetAddress.getByAddress(host, ipAddressToByteArray(ips[i]));
        }
        return addresses;
    }

   static byte[] ipAddressToByteArray(String ip) throws IllegalArgumentException{

        byte[] address = ip2bytes(ip);
        if (address != null) {
            return address;
        }
        throw new IllegalArgumentException(ip + ": INVALID_IP_ADDRESS");
    }

    public static byte[] ip2bytes(String ipAddr) {
        byte[] ret = new byte[4];
        try {
            String[] ipArr = ipAddr.split("\\.");
            ret[0] = (byte) (Integer.parseInt(ipArr[0]) & 0xFF);
            ret[1] = (byte) (Integer.parseInt(ipArr[1]) & 0xFF);
            ret[2] = (byte) (Integer.parseInt(ipArr[2]) & 0xFF);
            ret[3] = (byte) (Integer.parseInt(ipArr[3]) & 0xFF);
            return ret;
        } catch (Exception e) {
            throw new IllegalArgumentException(ipAddr + " is invalid IP");
        }

    }
}
```