# Java应用程序防反编译技术方案

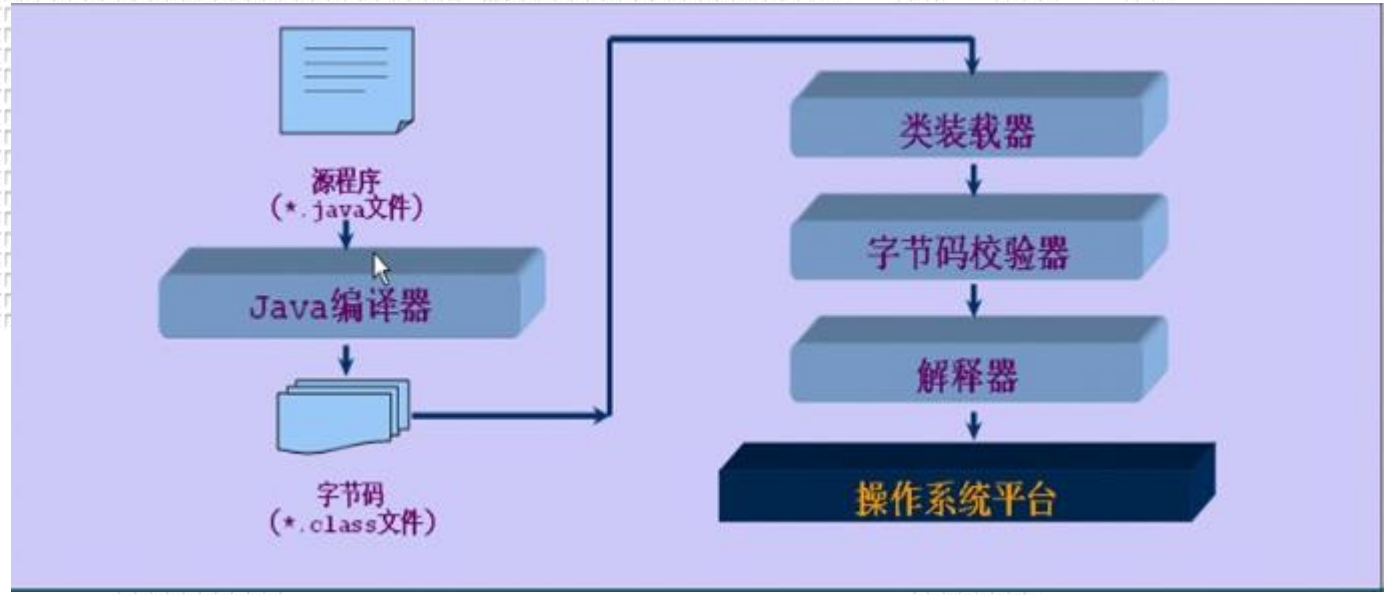
平台技术研究所平台研发组

王振洋

博智林机器人

2021年11月

# 方案背景 - Java语言的特点




| 开发理念 | 解释性语言 | 反编译容易 |
| --- | --- | --- |
| Java语言秉承共建共享的观念，提供jar和source包。 | 编译器会将Java代码变成“中间代码”，然后在Java虚拟机（Java Virtual Machine，JVM）上解释执行。 | 它是直接对其进行符号化、标记化的编译处理，根据class文件反向解析成原来的java文件。 |

# 内容简述

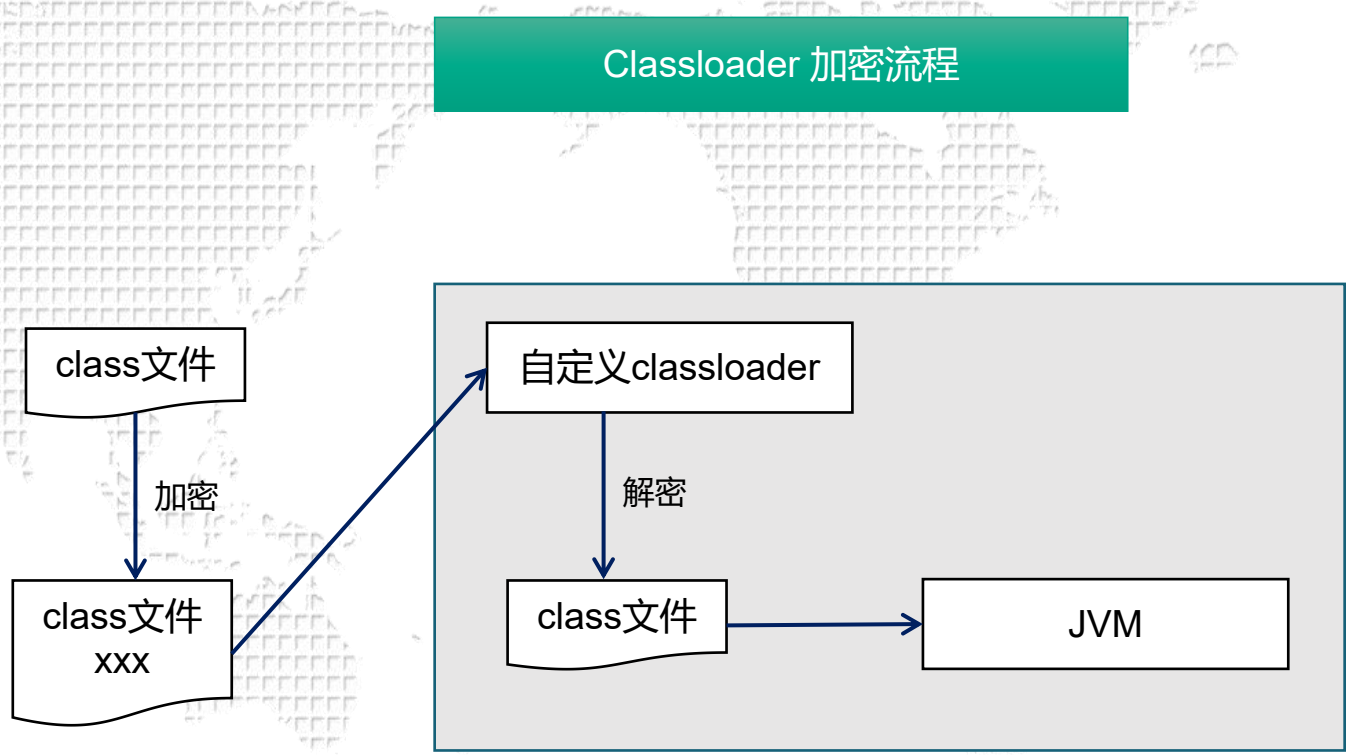
1 文件保护

2 服务化

3 组件化

4 JNA调用

# 方案1 Java文件的保护-class文件加密




springboot

JVMTI

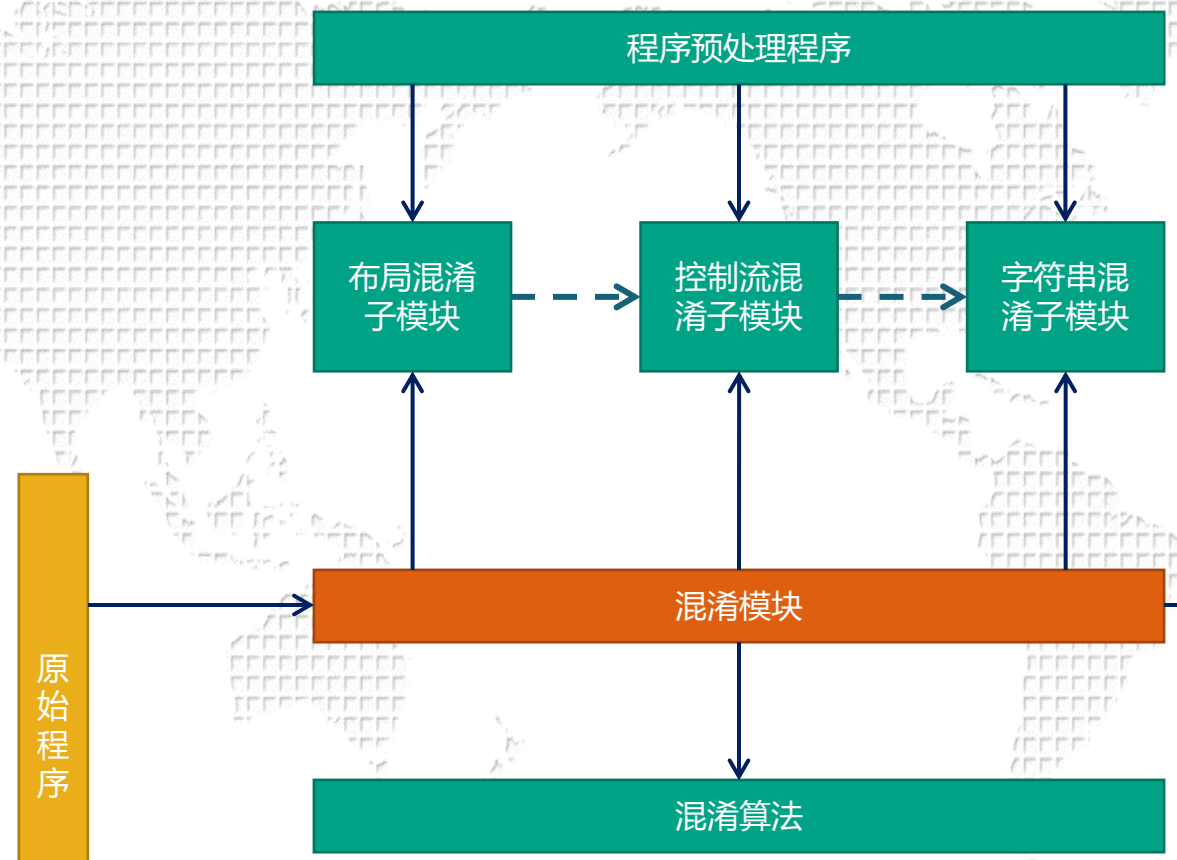
Virbox Protector

XJar

对class文件进行加密；然后用classloader进行解密。

# 方案1 class文件的保护-代码混淆



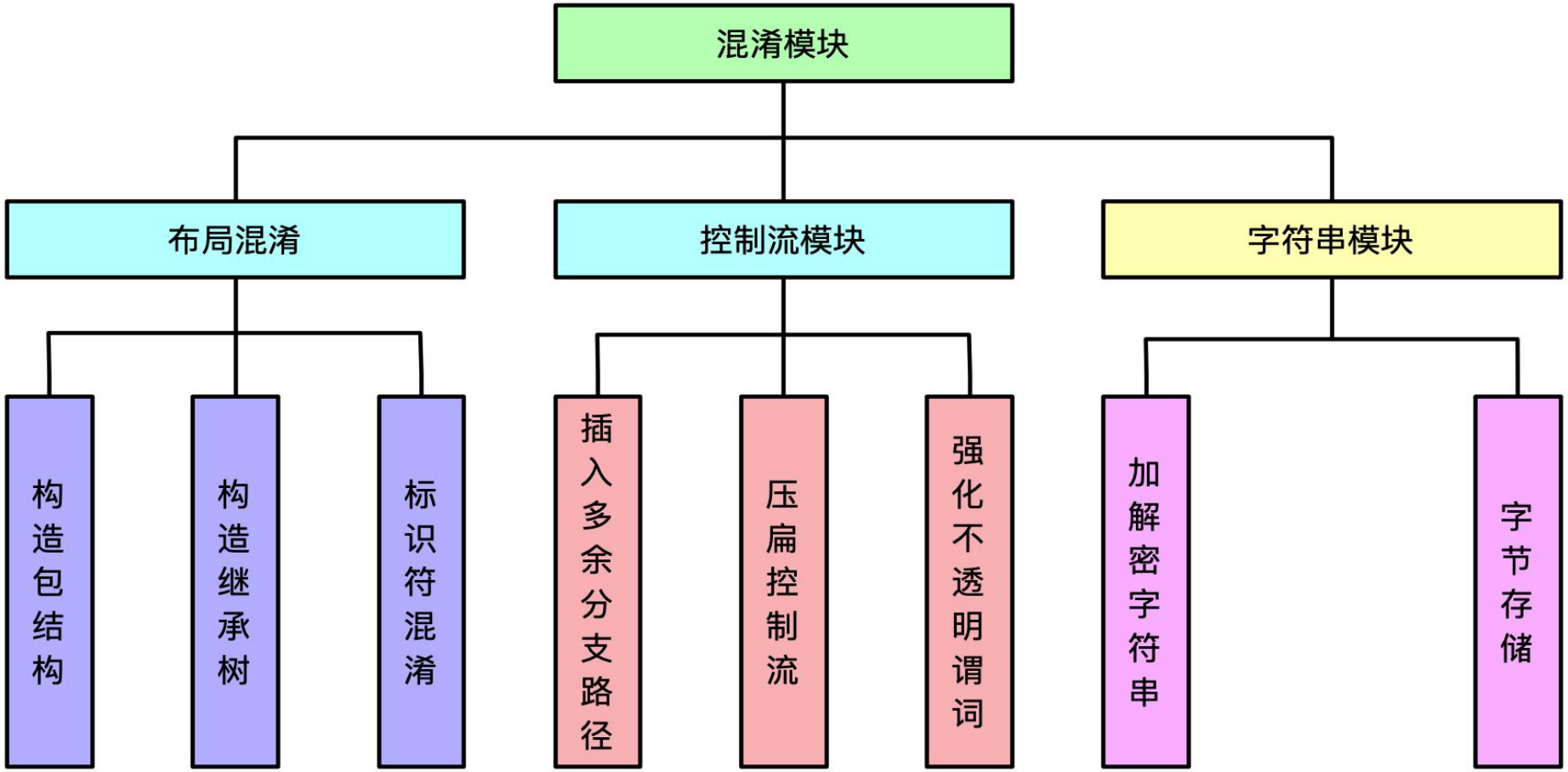



springboot

- Allatori
- JarProtector
- Zelix KlassMaster
- Cinnabar Canner

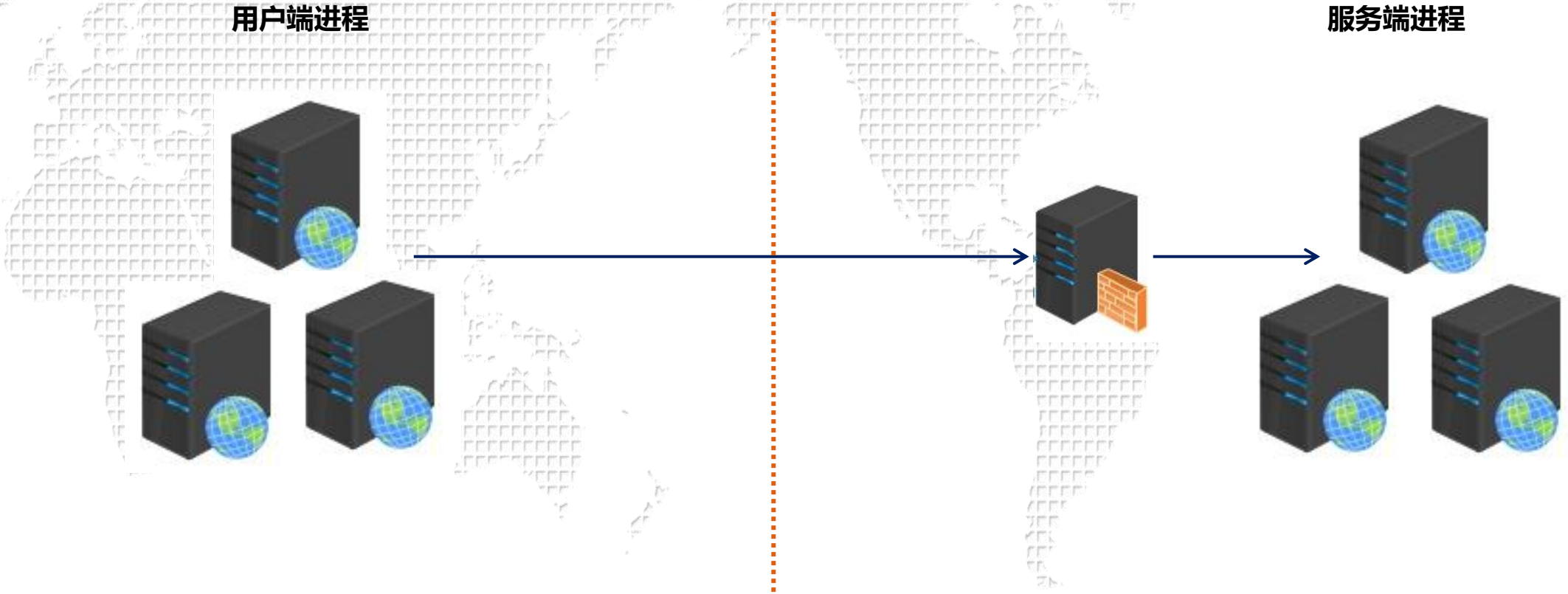
# 方案1 class文件的保护-代码混淆





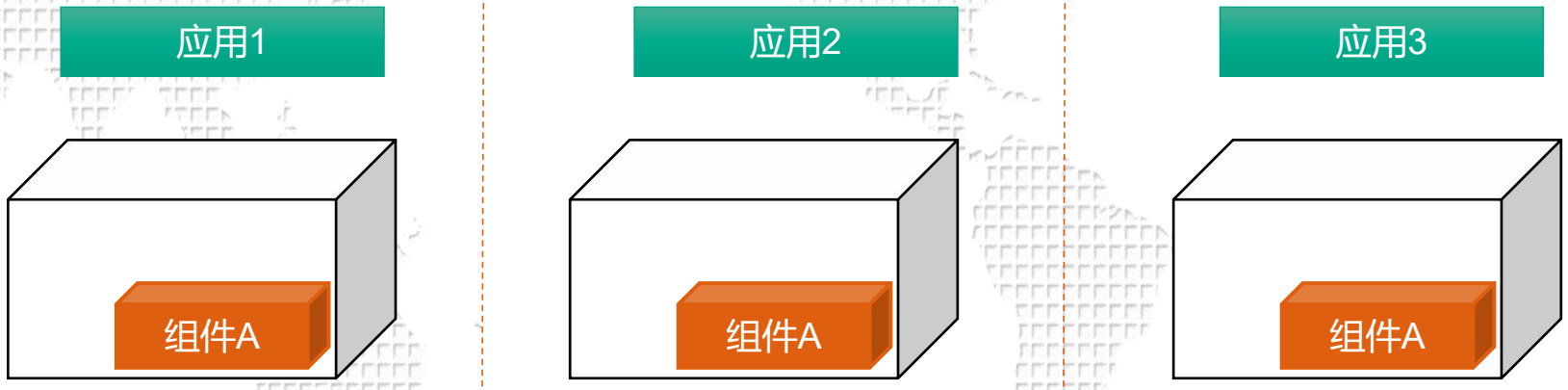
# 方案2 服务化--进程级远程调用





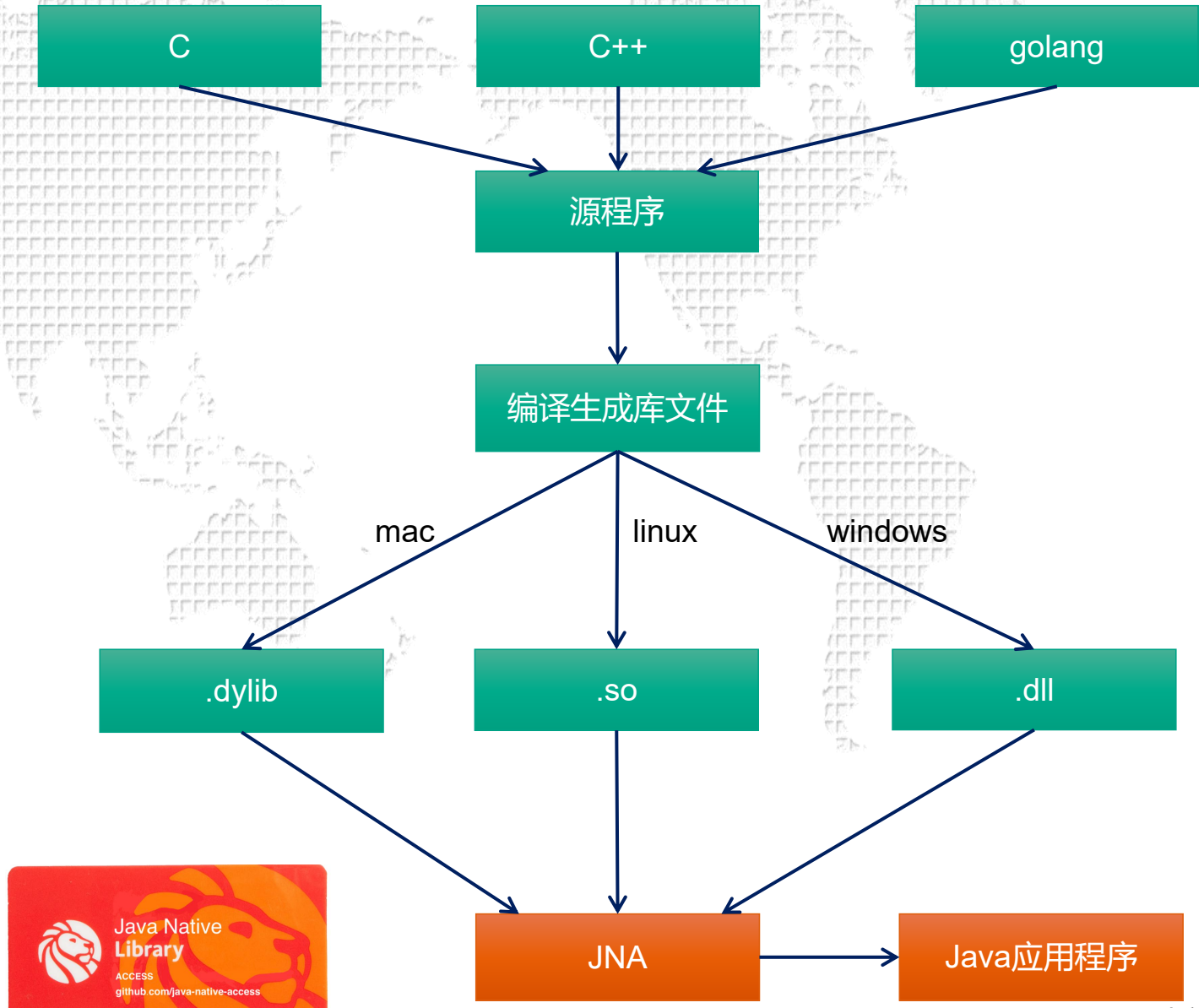

socket　rpc　https　...

# 方案3 组件化--进程级远程调用




适用于部分提供源码的场景。
二次开发或者需要扩展的场景。

# 方案4 JNA--更高级的组件化



# 方案4 JNA--更高级的组件化



| C Type | Native Representation | Java Type |
|---|---|---|
| char | 8-bit integer | byte |
| int | 32-bit integer | int |
| int | boolean flag | boolean |
| float | 32-bit floating point | float |
| double | 64-bit floating point | double |
| pointer (e.g. void*),<br>array | 32- or 64-bit pointer to memory (argument/return)<br>contiguous memory (struct member) | <P>[] (array of primitive type) |
| In addition to the above types, which are supported at the native layer, the JNA Java library automatically handles the following types. All but NativeMapped and NativeLong are converted to Pointer before being passed to the native layer. | | |
| const char* | NUL-terminated array (native encoding or jna.encoding) | String |
| struct*<br>struct | pointer to struct (argument or return) (or explicitly)<br>struct by value (member of struct) (or explicitly) | Structure |

http://java-native-access.github.io/jna/5.5.0/javadoc/overview-summary.html

# 方案4 JNA--更高级的组件化



```go
package main

import (
    "unsafe"
)
import "C"

//export Say
func Say(raw *C.char, size C.int) *C.char {
    data := C.GoBytes(unsafe.Pointer(raw), size)
    dataStr := "Hello," + string(data)
    return C.CString(dataStr)
}

func main(){
}
```

```java
public class GoEngineSayTest {

    public interface Awesome extends Library {
        String Say(byte[] raw, int len);
    }

    public static void main(String[] args) {
        String pwd = System.getProperty("user.dir");
        System.out.println("pwd : "+pwd);
        String lib = pwd + "/lib/lib-cqm-say";
        Awesome awesome = (Awesome) Native.loadLibrary(lib, Awesome.class);
        String str = "  -----world999999999";
        byte raw[] = str.getBytes();

        String res =awesome.Say(raw, raw.length);
        System.out.println(res);
    }
}
```

生成DLL命令：

go build -buildmode=c-shared -o bin/lib-cqm-say.dll cqm_say.go

# 技术路线比较



| 技术路线 | 优点 | 缺点 | 适用环境 | 其他 |
| --- | --- | --- | --- | --- |
| 加密class | 可以完全保护class不被反编译。 | 需要改在jdk/jre的classloader，自定义的classloader可能被反编译。 | 简单的应用程序。 | 增加了复杂性，同时引入了加密算法的安全问题 |
| 代码混淆 | 是一种成熟的反编译技术，可以找到较为完善的解决方案。 | 对混淆工具的依赖性很大。 | 适用于所有的情况。 | 是Java程序保护的基础。 |
| 服务化 | 完全保护class不被反编译。 | 需要安全机制保护接口的使用。 | 网络环境下，客户/服务器结构，或者分布式环境。 | 网络环境要求高。 |
| 组件化 | 关键的代码以特有的jar包封装，有利于统一进行反编译处理。 | 代码的管理较为分散，存在被反编译的概率更高。 | 本地部署；<br>适用于部分提供源码的场景。 | 本身并不能解决反编译的问题。 |
| JNA | 安全程度很高。 | 牺牲了跨平台特性，有时需要引入JNI技术，加大了系统的复杂性。 | 对关键算法或保密功能的保护。 | 通常需要对文件进行签名和验证。 |

# 参考文档

http://java-native-access.github.io/jna/5.5.0/javadoc/overview-summary.html

https://github.com/java-native-access/jna

https://github.com/vladimirvivien/go-cshared-examples

# 谢谢观看

# 阳光无畏，以终为始
# 价值创造，使命必达