HARBIN INSTITUTE OF TECHNOLOGY

Chapter 5: Reusability-Oriented Software Construction Approaches
5.1 Metrics, Morphology and External Observations of Reusability

Wang Zhongjie
rainy@hit.edu.cn

April 1, 2018

Objective of this lecture

- **Advantages and disadvantages of software reuse**

- **Construction for/with reuse**

- **Characteristics of generic reusable components**

- **Methods of developing portable application systems**

第3章介绍了软件构造的核心理论（ADT）与技术（OOP），其核心是保证代码质量、提高代码适应性和复用性。

本章面向一个重要的外部质量指标：可复用性——如何构造出可在不同应用中重复使用的软件模块/API？

5-1节探讨可复用软件的形态与特征，下一节学习"如何构造"

Outline

- **What is software reuse?**

- **How to measure "reusability"?**

- **Levels and morphology of reusable components**
  - Source code level reuse 源代码级别的复用
  - Module-level reuse: class/interface 模块级别的复用：类/抽象类/接口
  - Library-level: API/package 库级别的复用：API/包
  - System-level reuse: framework 系统级别的复用：框架

- **External observations of reusability**
  - Type Variation
  - Routine Grouping
  - Implementation Variation
  - Representation Independence
  - Factoring Out Common Behaviors

- **Summary**

1 What is Software Reuse?

Hardware is reused inherently
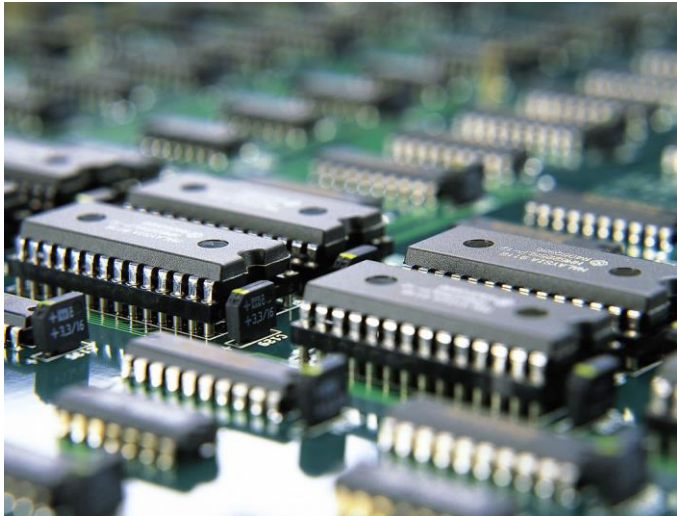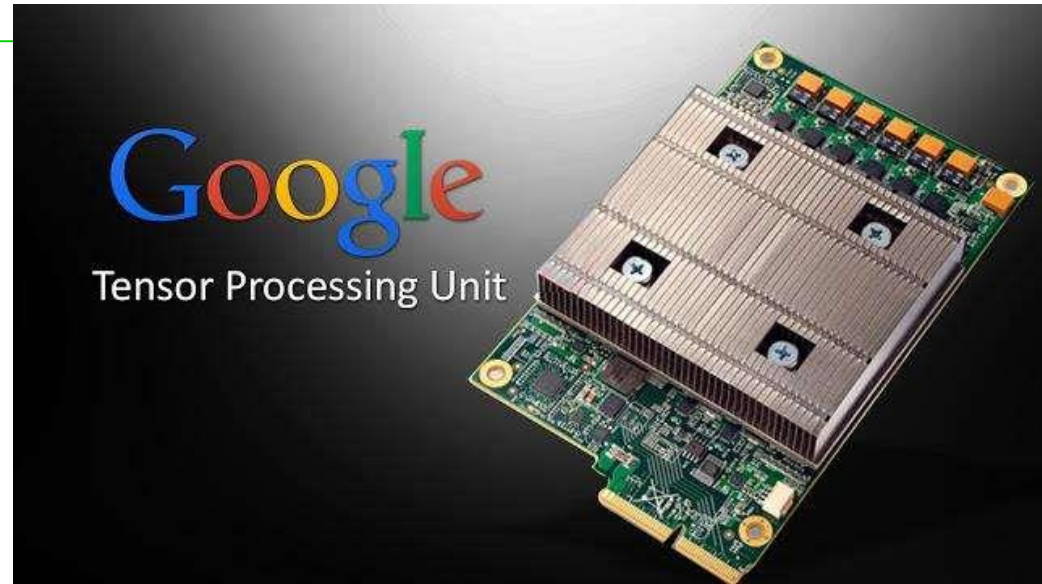
Software reuse

- **Software reuse** is the process of implementing or updating software systems using existing software components.

- **Two perspectives of software reuse**
  - Creation: creating reusable resources in a systematic way (programming for reuse 面向复用编程：开发出可复用的软件)
  - Use: reusing resources as building blocks for creating new systems (programming with reuse 基于复用编程：利用已有的可复用软件搭建应用系统)

- **Why reuse?**
  - "The drive to create reusable rather than transitory artifacts has aesthetic and intellectual as well as economic motivations and is part of man's desire for immortality.
  - It distinguishes man from other creatures and civilized from primitive societies" (Wegner, 1989).

Recall Lab2

- 在**Lab2**中：你开发了一个基于泛型的抽象接口**Graph<L>**，定义了支持图结构的**ADT**

- 针对该**ADT**，用两种不同的**Rep**，开发了两个不同的实现
 **ConcreteVertexGraph<L>**和**ConcreteEdgeGraph<L>**

> Programming for reuse
> 面向复用编程：开发出可复用的软件

- 进而，你利用该**ADT**及其两个实现，完成了三个应用的开发：

  – Poetic Walks

  – Friendship Social Network

  – Transportation Network

> Programming with reuse
> 基于复用编程：利用已有的可复用软
> 件搭建应用系统

Why reuse?

- **Reuse is cost-effective and with timeliness 降低成本和开发时间**

  – Increases software productivity by shortening software production cycle time (software developed faster and with fewer people)

  – Does not waste resources to needlessly "reinvent-the-wheel"

  – Reduces cost in maintenance (better quality, more reliable and efficient software can be produced)

- **Reuse produces reliable software 经过充分测试，可靠、稳定**

  – Reusing functionality that has been around for a while and is debugged is a foundation for building on stable subsystems

- **Reuse yields standardization 标准化，在不同应用中保持一致**

  – Reuse of GUI libraries produces common look-and-feel in applications.

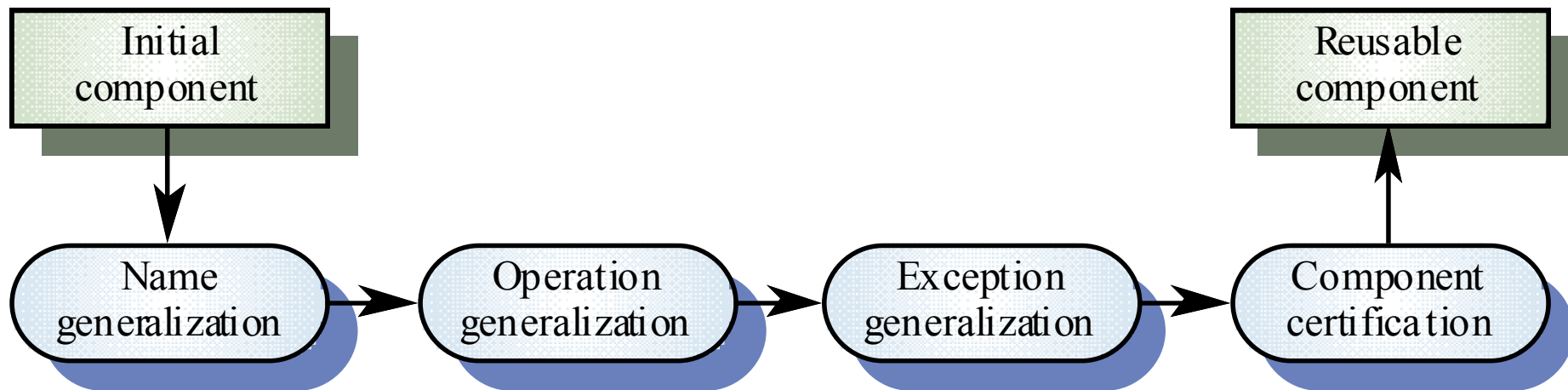 – Consistency with regular, coherent design.
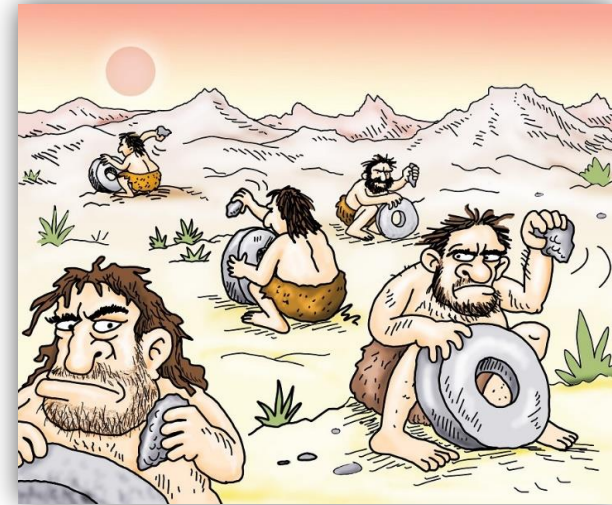
Reuse costs

- **Reusable components** should be designed and built in a *clearly* defined, *open* way, with *concise* interface specifications, *understandable* documentation, and an eye towards *future use*. 做到这些，需要代价

- **Reuse is costly:** it involves spans organizational, technical, and process changes, as well as the cost of tools to support those changes, and the cost of training people on the new tools and changes. 不仅 program for reuse代价高，program with reuse代价也高
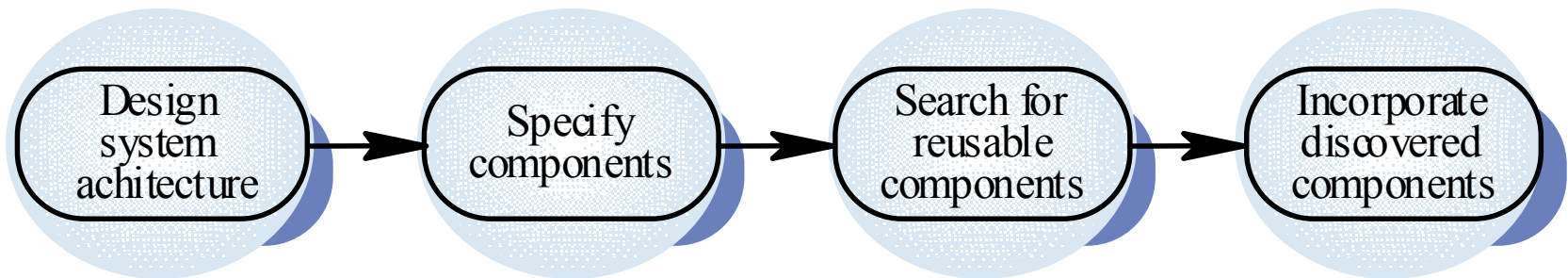
Cost

Without reuse

With reuse

Products
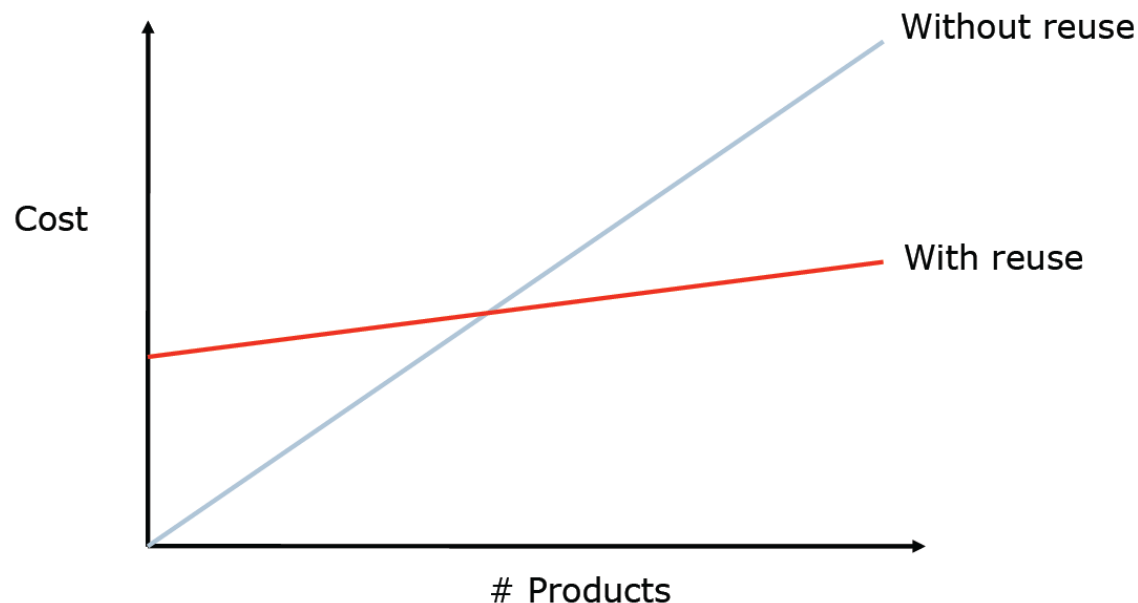
Development for reuse: 开发可复用的软件

- **The development cost of reusable components is higher than the cost of specific equivalents. This extra reusability enhancement cost should be an organization rather than a project cost. 开发成本高于一般软件的成本：要有足够高的适应性**

- **Generic components may be less space-efficient and may have longer execution times than their specific equivalents. 性能差些：针对更普适场景，缺少足够的针对性**

```
Initial                                           Reusable
component                                          component

   │                                                  ▲
   ▼                                                  │
┌──────────┐    ┌──────────┐    ┌──────────┐    ┌──────────┐
│   Name   │──▶ │Operation │──▶ │Exception │──▶ │Component │
│generalization│ │generalization│ │generalization│ │certification│
└──────────┘    └──────────┘    └──────────┘    └──────────┘
```

Development with reuse: 使用已有软件进行开发

- **Component management tools, such as repositories, for architectures, designs, documentation, and code must be developed and maintained. 可复用软件库，对其进行有效的管理**

```
Design
system
achitecture   →   Specify
                  components   →   Search for
                                   reusable
                                   components   →   Incorporate
                                                    discovered
                                                    components
```

- **A key issue: adaptation 往往无法拿来就用，需要适配**

  - Extra functionality may have to be added to a component. When this has been added, the new component may be made available for reuse.

  - Unneeded functionality may be removed from a component to improve its performance or reduce its space requirements

  - The implementation of some component operations may have to be modified.

2 How to measure "reusability"?

Measure resuability

- **How frequently can a software asset be reused in different application scenarios?** 复用的机会有多频繁？复用的场合有多少？
  - The more chance an asset is used, the higher reusability it has.
  - Write once, reuse multiple times.

- **How much are paid for reusing this asset?** 复用的代价有多大？
  - Cost to buy the asset and other mandatory libraries 搜索、获取
  - Cost for adapting and extending it 适配、扩展
  - Cost for instantiating it 实例化
  - Cost for changing other parts of the system that interact with it 与软件其他部分的互连

Reusability

- **Reusability implies some explicit management of <u>build</u>, <u>packaging</u>, <u>distribution</u>, <u>installation</u>, <u>configuration</u>, <u>deployment</u>, <u>maintenance</u> and <u>upgrade</u> issues.**

- **A software asset with high reusability should:**
  - Brief (small size) and Simple (low complexity)   小、简单
  - Portable and Standard Compliance   与标准兼容
  - Adaptable and Flexible   灵活可变
  - Extensibility   可扩展
  - Generic and Parameterization   泛型、参数化
  - Modularity   模块化
  - Localization of volatile (changeable) design assumptions 变化的局部性
  - Stability under changing requirements   稳定
  - Rich documentation   丰富的文档和帮助

Reusable Libraries and APIs in JDK
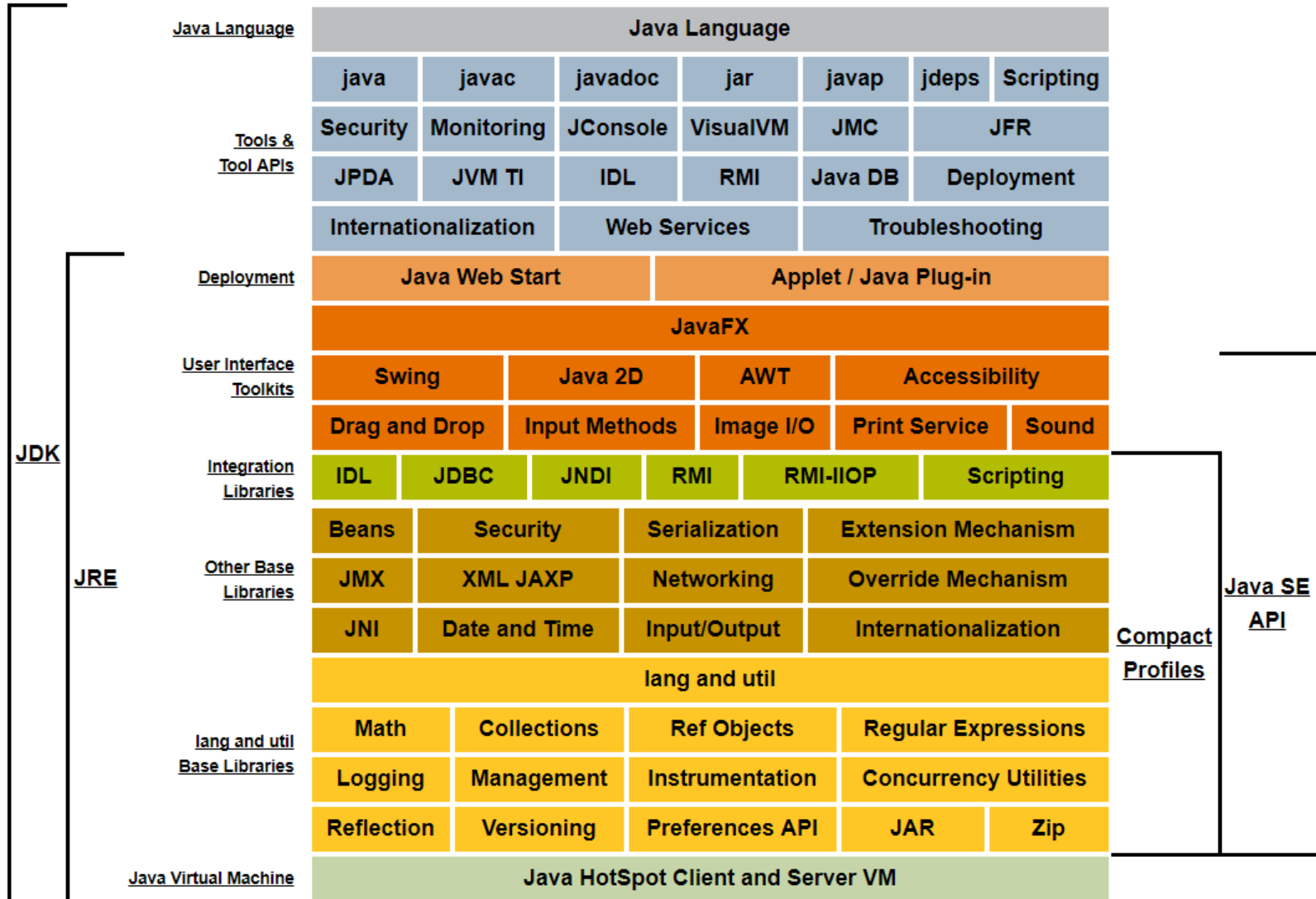
Rich third-party libraries and APIs in Java

Central Repository
http://central.maven.org/maven2/

URL	http://central.maven.org/maven2/
Jars	2,467,181 indexed jars

Published Jars by Year	
2018	31,819
2017	709,421
2016	539,245
2015	362,410
2014	229,885
2013	170,787
2012	126,711
2011	106,285
2010	86,191
2009	49,537
2008	25,685
2007	12,478
2006	7,563
2005	9,067
2004	1

Home » log4j » log4j » 1.2.17

Apache Log4j » 1.2.17
Apache Log4j 1.2

License	Apache 2.0
Categories	Logging Frameworks
Organization	Apache Software Foundation
HomePage	http://logging.apache.org/log4j/1.2/
Date	(May 26, 2012)
Files	pom (21 KB) bundle (478 KB) View All
Repositories	Central Apache Releases Redhat GA Sonatype Releases Spring Plugins
Used By	**11,493 artifacts**

Maven | Gradle | SBT | Ivy | Grape | Leiningen | Buildr

```
<!-- https://mvnrepository.com/artifact/log4j/log4j -->
<dependency>
    <groupId>log4j</groupId>
    <artifactId>log4j</artifactId>
    <version>1.2.17</version>
</dependency>
```

☑ Include comment with link to declaration

Rich RubyGems

寻找、安装以及发布
RubyGems

Search Gems...

Advanced Search →

18,947,733,215
总下载次数

安装 RubyG

Stats

TOTAL GEMS
139,663

TOTAL USERS
118,801

下载总次数
18,947,744,300

ALL TIME MOST DOWNLOADED

DISPLAYING GEMS **1 - 10** OF **100** IN TOTAL

Gem	Downloads
bundler	217,398,175
multi_json	189,349,841
rake	186,861,120
rack	185,815,759
json	178,094,053
mime-types	173,826,045
activesupport	165,309,018
rspec-core	164,624,569
diff-lcs	164,590,605
rspec-expectations	163,709,140
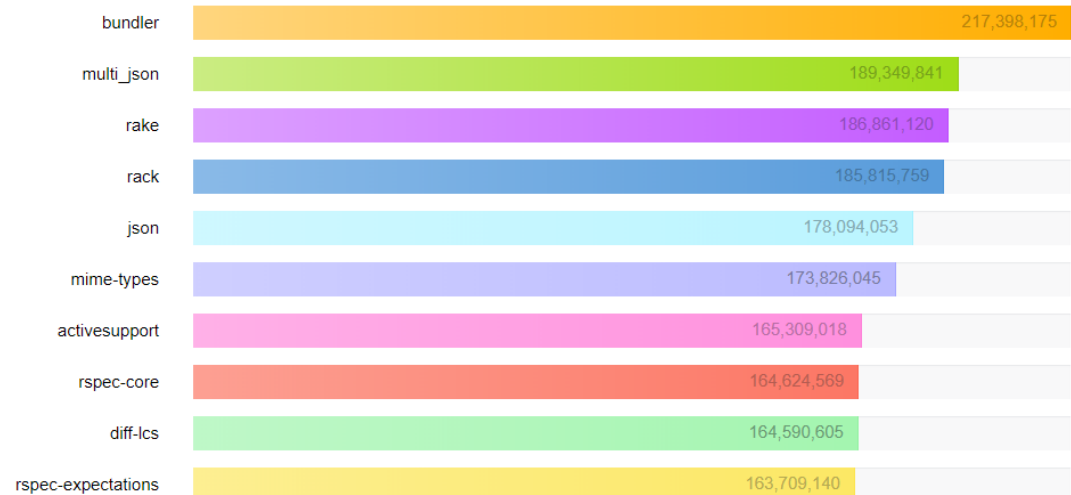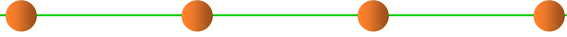
3 Levels and morphology of reusable components

Levels of Reuse

- **A reusable component may be code 最主要的复用是在代码层面**
  - Most prevalent: what most programmers relate with reuse

- **But benefits result from a broader and higher-level view of what can be reused. 但软件构造过程中的任何实体都可能被复用**
  - Requirements 需求
  - Design and specifications 设计/规约spec
  - Data 数据
  - Test cases 测试用例
  - Documentation 文档

What we concern in this lecture

- **Source code level: methods, statements, etc**

- **Module level: class and interface**

- **Library level: API**

  – Java Library, `.jar`

- **Architecture level: framework 框架**

Types of Code Reuse

- **White box reuse** 白盒复用：源代码可见，可修改和扩展
  - Reuse of code when code itself is available. Usually requires some kind of modification or adaptation 复制已有代码当正在开发的系统，进行修改
  - Pro: You can customize the module to fit the specific situation, this allows reuse in more situations 可定制化程度高
  - Con: You now own the customized result, so it adds to your code complexity. You requires intrinsic knowledge on component internals. 对其修改增加了软件的复杂度，且需要对其内部充分的了解

- **Black box reuse** 黑盒复用：源代码不可见，不能修改
  - Reuse in the form of combining existing code by providing some "glue", but without having to change the code itself - usually because you do not have access to the code 只能通过API接口来使用，无法修改代码
  - Pro: Simplicity and Cleanliness 简单，清晰
  - Con: Many times it is just not possible 适应性差些

Formats for reusable component distribution

- **Forms:**
  - Source code
  - Package such as .jar, .gem, .dll,

以及最重要的：自己日积月累出来的可复用代码库

- **Sources of reusable software components:**
  - Internal (corporate) code libraries   组织的内部代码库（Guava）
  - Third party libraries   第三方提供的库（Apache）
  - Built-in language libraries   语言自身提供的库（JDK）
  - Code samples from tutorials, examples, books, etc. 代码示例
  - Local code guru or knowledgeable colleague   来自同事
  - Existing system code   已有系统内的代码
  - Open source products (be sure to follow any licensing agreements) 开源软件的代码

(1) Source code reuse

Reusing Code – Lowest Level

- **Copy/paste parts/all into your program**

- **Maintenance problem**
  - Need to correct code in multiple places
  - Too much code to work with (lots of versions)
- **High risk of error during process**
- **May require knowledge about how the used software works**
- **Requires access to source code**

- 相关研究1：如何从互联网上快速找到需要的代码片段？
- 反向研究：如何从源代码中检测出克隆代码(clone code)？

Example code search: grepcode.com

GitHub code search: github.com/search

Searchcode: searchcode.com

(2) Module-level reuse: class/interface

Inheritance

Use

Composition/aggregation

Delegation/association

Reusing classes

- **A class is an atomic unit of code reuse**
  - Source code not necessary, class file or jar/zip
  - Just need to include in the classpath
  - Can use javap tool to get a class's public method headers

- **Documentation very important (Java API)**

- **Encapsulation helps reuse**

- **Less code to manage**

- **Versioning, backwards-compatibility still problem**

- **Need to package related classes together** -- Static Linking

Approaches of reusing a class: inheritance

- **Java provides a way of code reuse named** `Inheritance`
  - Classes extend the properties/behavior of existing classes
  - In addition, they might `override` existing behavior

- **No need to put dummy methods that just forward or delegate work**

- **Captures the real world better**

- **Usually need to design inheritance hierarchy before implementation**

- **Cannot cancel out properties or methods, so must be careful not to overdo it**

Approaches of reusing a class: delegation

- `Delegation` **is simply when one object relies on another object for some subset of its functionality** (one entity passing something to another entity)
  - e.g. the `Sorter` is delegating functionality to some `Comparator`
- **Judicious delegation enables code reuse**
  - `Sorter` can be reused with arbitrary sort orders
  - `Comparators` can be reused with arbitrary client code that needs to compare integers
- **Explicit delegation:** passing the sending object to the receiving object
- **Implicit delegation:** by the member lookup rules of the language

- **Delegation** can be described as a low level mechanism for sharing code and data between entities.

(3) Library-level reuse: API/Package

Libraries

- **Library: A set of classes and methods (APIs) that provide reusable functionality**

Math

Collections

Graphs

I/O

Library

Swing
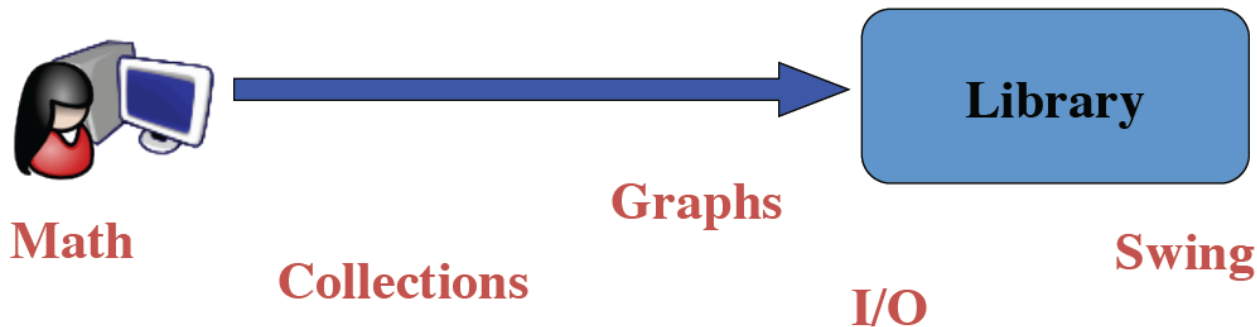
Framework

- **Framework: Reusable skeleton code that can be customized into an application**

- **Framework calls back into client code**

  - The Hollywood principle: "Don't call us. We'll call you."

your code

Eclipse   Firefox   Framework   Swing

Applet   Spring
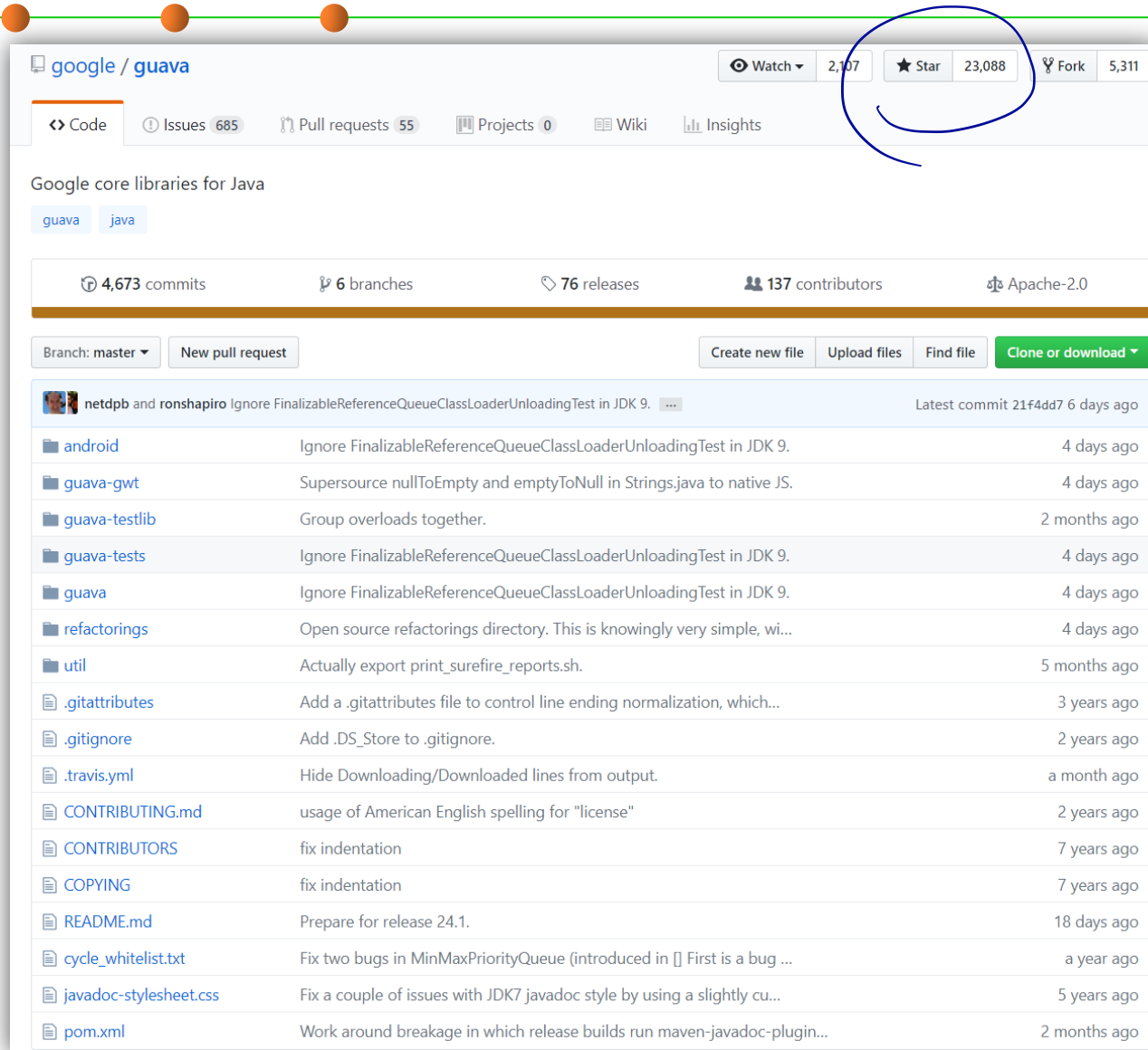
General distinction: Library vs. framework
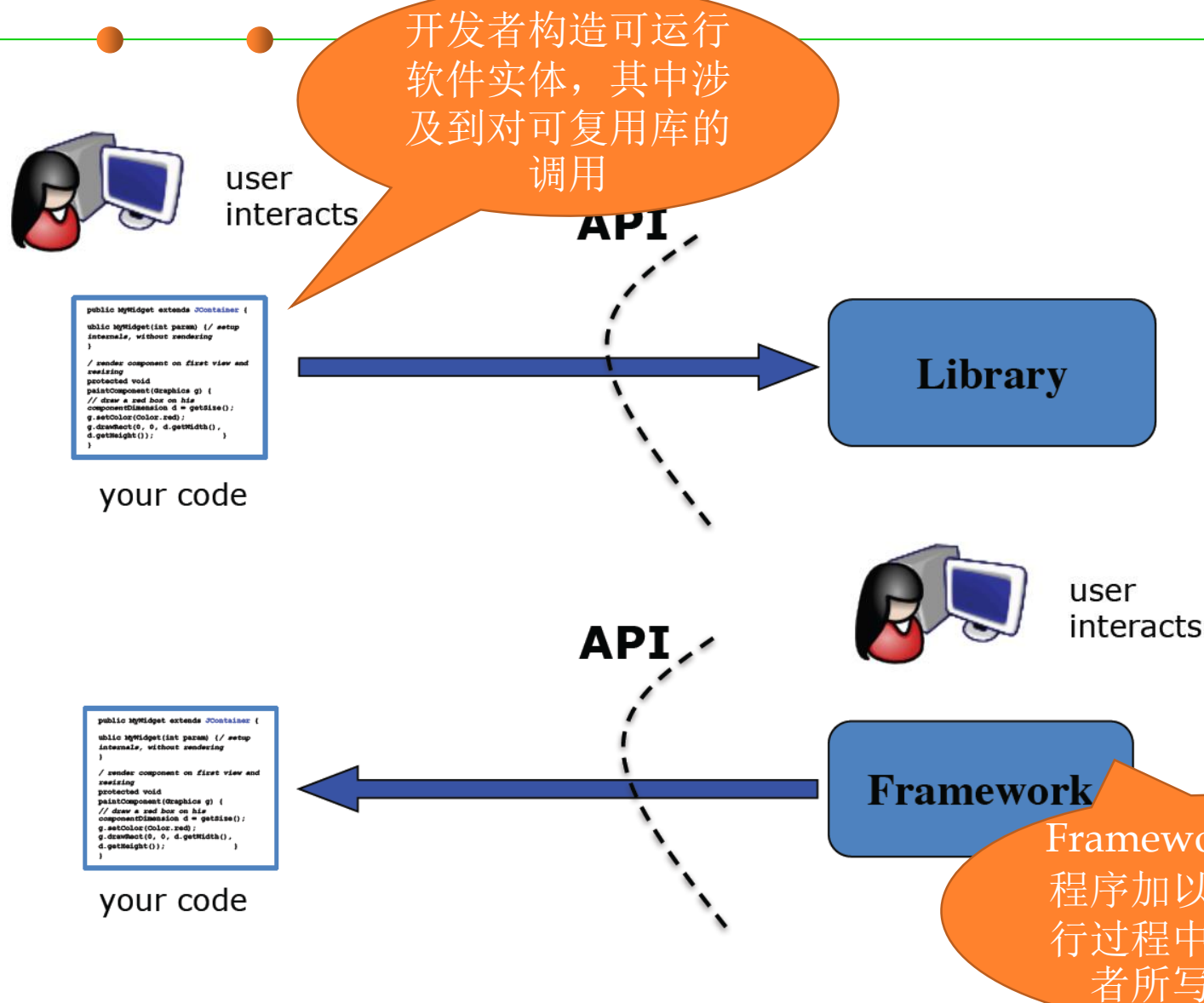
Characteristics of a good API

- **Easy to learn**

- **Easy to use, even without documentation**

- **Hard to misuse**

- **Easy to read and maintain code that uses it**

- **Sufficiently powerful to satisfy requirements**

- **Easy to evolve**

- **Appropriate to audience**

Guava: Google core libraries for Java

Apache Commons

- **Apache Commons is an Apache project focused on all aspects of reusable Java components.**

  - https://commons.apache.org

  - https://github.com/apache/commons-*

API on Web/Internet
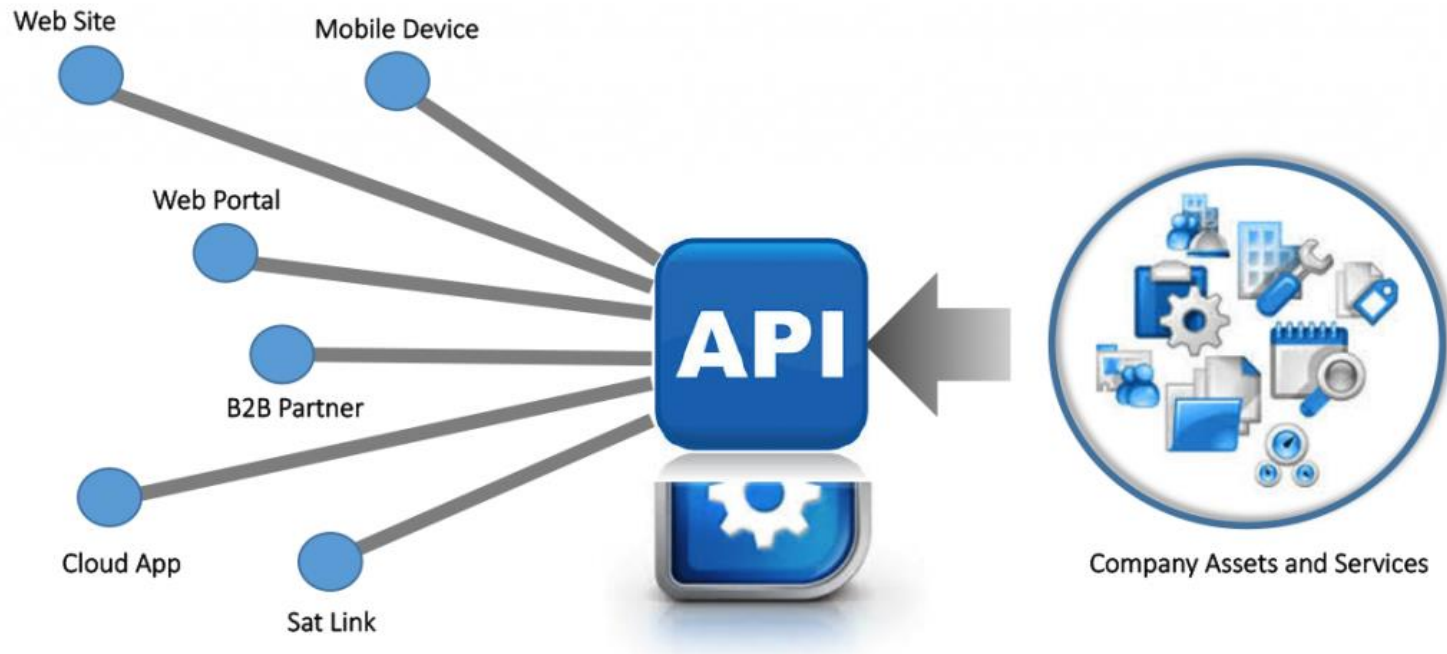
从现在开始，除非实验手册中明确禁止，你可以使用任何公开发布的可复用库完成Labs中的任何要求

并且，大力提倡！

(4) System-level reuse: Framework

Application Frameworks

- **Frameworks are sub-system design containing a collection of abstract and concrete classes along with interfaces between each class 框架：一组具体类、抽象类、及其之间的连接关系**

  - 只有"骨架"，没有"血肉"

- **A framework is an abstraction in which software providing generic functionality can be selectively changed by additional user-written code, thus providing application-specific software. 开发者根据 framework的规约，填充自己的代码进去，形成完整系统**

Framework

Application Components
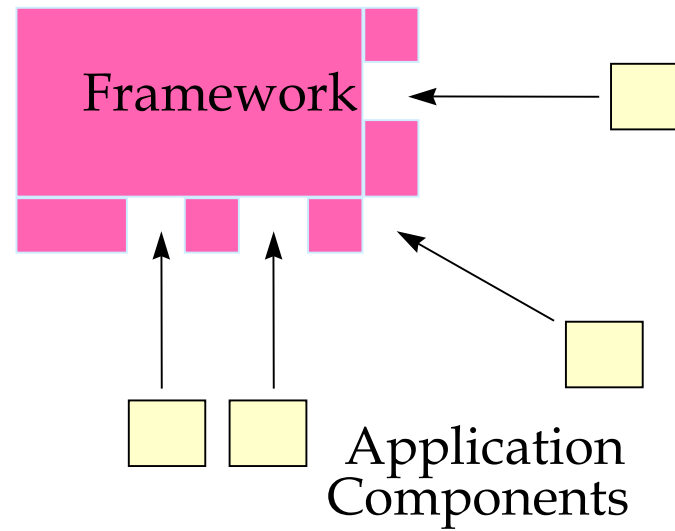
Application Frameworks

- **Reusability leverages of the application domain knowledge and prior effort of experienced developers 领域知识的复用**

  - Data processing, GUI, etc

  - 将framework看作是更大规模的API复用，除了提供可复用的API，还将这些模块之间的关系都确定下来，形成了整体应用的领域复用

Framework作为主程序加以执行，执行过程中调用开发者所写的程序
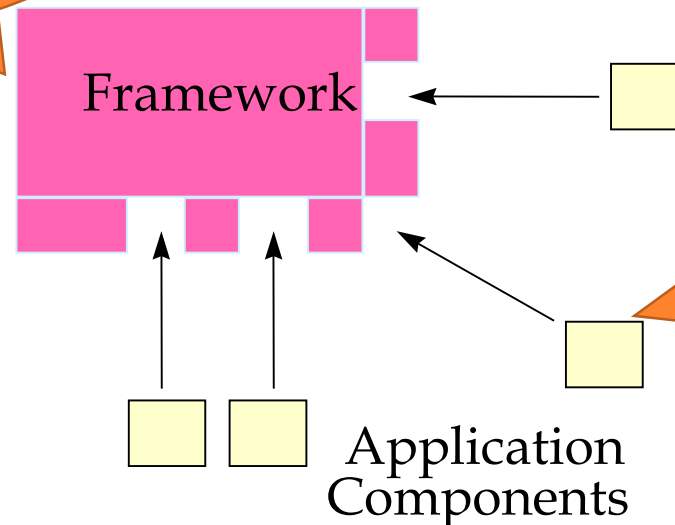
Framework

开发者根据Framework预留的接口所写的程序

Application Components

Frameworks: domain-level reuse 领域复用

- **A system is implemented by adding components to fill in missing design elements and by instantiating the abstract classes 开发者：增加新代码、对抽象类进行具体化**

  - Extend the framework usually by selective overriding; or programmers can add specialized user code to provide specific functionality --- defining concrete classes that inherit operations from abstract class ancestors

  - Hook methods, which are overwritten by the application to extend the framework. Hook methods systematically decouple the interfaces and behaviors of an application domain from the variations required by an application in a particular context.

  - Inversion of control: Unlike in libraries or in standard user applications, the control flow is not dictated by the caller, but by the framework.

  - Non-modifiable framework code: The framework code is not supposed to be modified, while accepting user-implemented extensions. In other words, users can extend the framework, but should not modify its code.

控制反转(Inverse of Control)

- 由第三方的容器来控制对象之间的依赖关系，而非传统实现中由代码直接操控。

- 控制权由代码中转到了外部容器，带来的好处就是降低了对象之间的依赖程度，提高灵活性和可维护性。

Class B

Class A
希望调用B的方法

请求B的实例

创建B的实例

返回B的实例

容器

配置文件
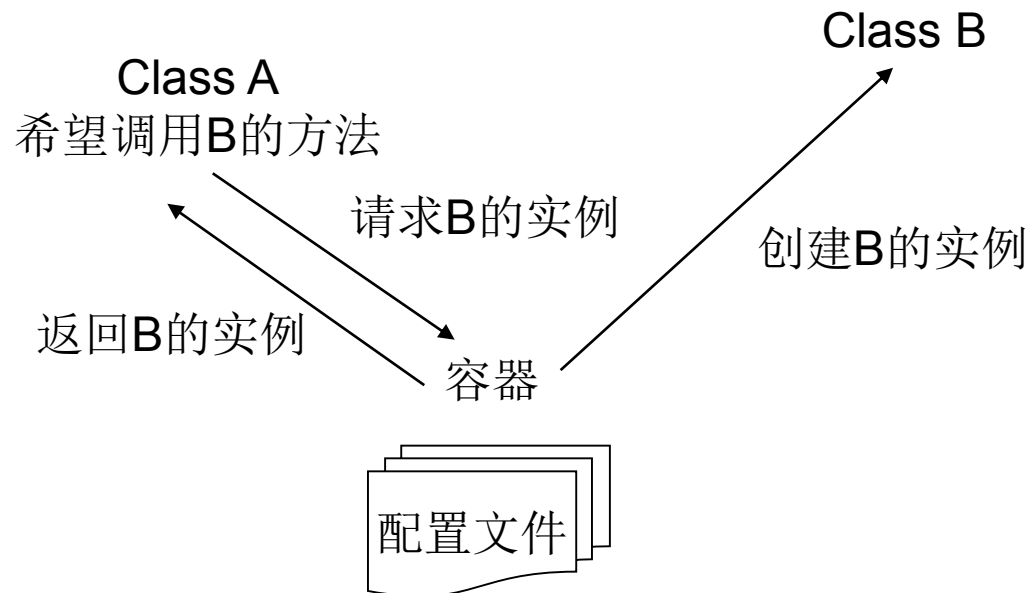
Framework Design

- **Frameworks differ from applications**

  – The level of abstraction is different as frameworks provide a solution for a family of related problems, rather than a single one.

  – To accommodate the family of problems, the framework is incomplete, incorporating hot spots and hooks to allow customization

- **Frameworks can be classified by the techniques used to extend them.**

  – Whitebox frameworks

  – Blackbox frameworks

White-box and Black-Box Frameworks

- **Whitebox frameworks:**
  - Extensibility achieved through inheritance and dynamic binding.
  - Existing functionality is extended by subclassing framework base classes and overriding predefined hook methods
  - Often design patterns such as the template method pattern are used to override the hook methods.

- **Blackbox frameworks**
  - Extensibility achieved by defining interfaces for components that can be plugged into the framework.
  - Existing functionality is reused by defining components that conform to a particular interface
  - These components are integrated with the framework via delegation.

Class libraries vs. Frameworks

- **Class Libraries:**
  - Less domain specific
  - Provide a smaller scope of reuse.
  - Class libraries are passive; no constraint on control flow.

- **Framework:**
  - Classes cooperate for a family of related applications.
  - Frameworks are active; affect the flow of control.

- **In practice, developers often use both:**
  - Frameworks often use class libraries internally to simplify the development of the framework.
  - Framework event handlers use class libraries to perform basic tasks (e.g. string processing, file management, numerical analysis….)

Components vs. Frameworks

- **Components**
  - Self-contained instances of classes
  - Plugged together to form complete applications.
  - Blackbox that defines a cohesive set of operations,
  - Can be used based on the syntax and semantics of the interface.
  - Components can even be reused on the binary code level.
    - The advantage is that applications do not always have to be recompiled when components change.

- **Frameworks:**
  - Often used to develop components
  - Components are often plugged into blackbox frameworks.

4 External observations of reusability

External observations of reusability

- **Type Variation**　　　　　　　类型可变

- **Routine Grouping**　　　　　　功能分组

- **Implementation Variation**　　实现可变

- **Representation Independence**　表示独立

- **Factoring Out Common Behaviors**　共性抽取

Type Variation

- **Reusable components should be type-parameterized so that they can adapt to different data types (input, computation, and output);**

  - A reusable module should be applicable to many different types of element, without requiring developers to perform manual changes to the software text.

  - In other words, we need a facility for describing type-parameterized modules, also known more concisely as generic modules.

- **Genericity: reusable components should be generic.**

- 类型可变(泛型)：适应不同的类型，且满足LSP

Implementation Variation

- **In practice a wide variety of applicable data structures and algorithms.**

- **Such variety indeed that we cannot expect a single module to take care of all possibilities; it would be enormous.**

- **We will need a family of modules to cover all the different implementations.**

- 实现可变：**ADT**有多种不同的实现，提供不同的**representations**和 **abstract funtion**，但具有同样的**specification (pre-condition, post-condition, invariants)**，从而可以适应不同的应用场景

Routine Grouping

- **A self-sufficient reusable module would need to include a set of routines, one for each of the operations.**

- **Completeness**

- 提供完备的细粒度操作，保证功能的完整性，不同场景下复用不同的操作(及其组合)

Representation Independence

- **A general form of reusable module should enable clients to specify an operation without knowing how it is implemented.**

- Representation Independence is an extension of the rule of Information Hiding, essential for smooth development of large systems: implementation decisions will often change, and clients should be protected. 内部实现可能会经常变化，但客户端不应受到影响。

- Representation Independence reflects the client's view of reusability — the ability to ignore internal implementation details and variants

- 表示独立性、信息隐藏

Factoring Out Common Behaviors

- **Factoring Out Common Behaviors**, reflects the view of the supplier and, more generally, the view of developers of reusable classes.

- **The goal will be to take advantage of any commonality that may exist within a family or sub-family of implementations.**

- 将共同的行为（共性）抽象出来，形成可复用实体

- The variety of implementations available in certain problem areas will usually demand, as noted, a solution based on a family of modules.

- Each of these categories covers many variants, but it is usually possible to find significant commonality between these variants.

Summary

Summary

- **What is software reuse?**

- **How to measure "reusability"?**

- **Levels and morphology of reusable components**
  - Source code level reuse
  - Module-level reuse: class/interface
  - Library-level: API/package
  - System-level reuse: framework

- **External observations of reusability**
  - Type Variation
  - Routine Grouping
  - Implementation Variation
  - Representation Independence
  - Factoring Out Common Behaviors

The end

April 1, 2018