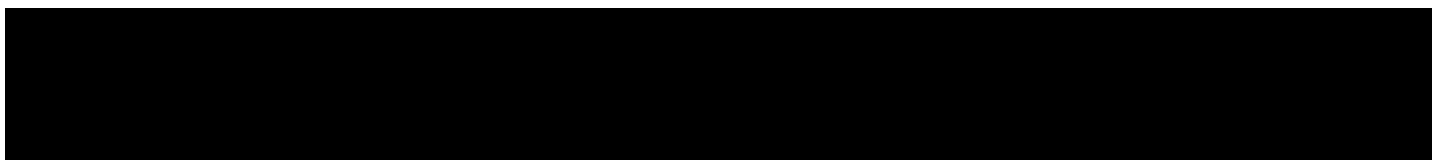


## 3. 命名



### 本章主要参考文献



- Jack B. Dennis. Segmentation and the design of multiprogrammed computer systems. Journal of the ACM 12, 4 (October 1965), 589–602.
- Robert S. Fabry. Capability-based addressing. Communications of the ACM 17, 7 ( July 1974), 403–412.
- Paul J. Leach, Bernard L. Stumpf, James A. Hamilton, and Paul H. Levine. UIDs as internal names in a distributed file system. In ACM SIGACT–SIGOPS Symposium on Principles of Distributed Computing, Ottawa, Ontario (August 18–20, 1982), 34–41.

# 回顾：抽象与命名

## ● 设计基本的计算机系统

1. 计算机系统的3个基本抽象：解释器、存储和通信，分析与举例
2. 计算机系统的分层抽象，举例与分析（总线、文件系统）
3. 命名
  - 用命名连接抽象模块
4. 实例
  - UNIX文件系统的分层抽象构造

### 提问：

1. UNIX文件系统能否减少分层且不影响功能？
2. 如果可以，会带来什么问题？

## ● 本次课程：专题讨论“命名”技术的若干问题

# 回顾：名称

## ● 3个全球唯一上下文的命名体系

- DNS域名
- IP地址
- Mac地址

哪个名称移动了位置会失效？

哪个名称更新后不一定马上看到？

哪个名称用户一般很少知道？

分别面临什么问题？

## ● 2个动态上下文的命名体系

- 程序设计语言的变量名
- 文件系统的文件名

哪个名称有7个以上的间接层？

哪个名称的生存期多种多样？

分别面临什么问题？

为什么它们各自不同？专题讨论

# 回顾：名称

- 名称是什么？

OOP为例

- 使外部观察者能够标识事物的术语
- 名称标识1组、1类或1个事物，或唯一或在上下文（context）中
- 被名称标识的事物称为指称对象或指示物（referent）

- 命名（naming）即对名称的创建和使用

- 命名支持间接，从而支持延迟绑定，易于替换，支持共享

SO为例

# 回顾：名称解析与比较

- 上下文：

- 默认、显式

shell运行命令

- 各种解析方式：

- 查表、递归、多重

链接器寻找库函数

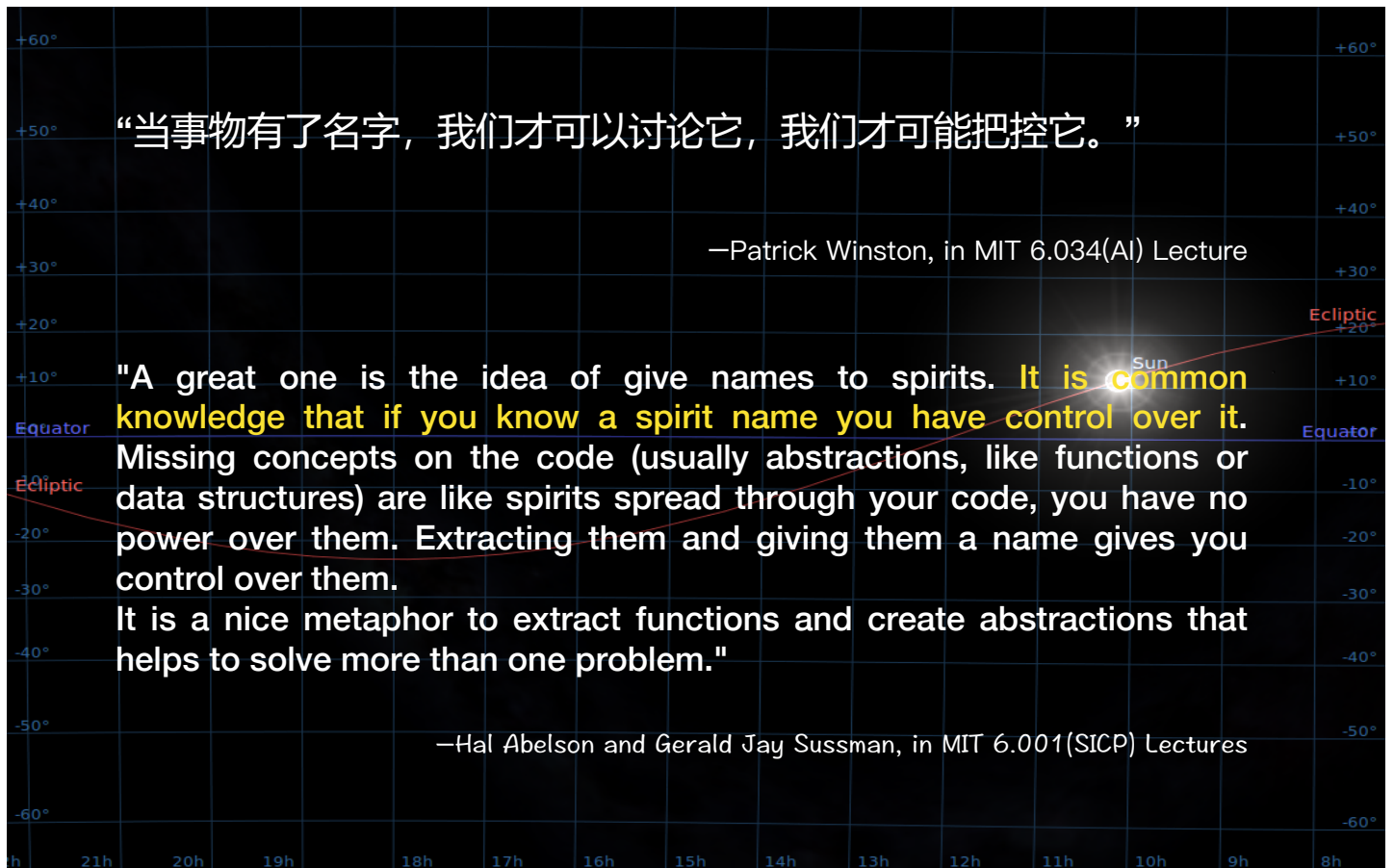
- 名称的相等性：

- 名相等、指示物相同、指示物相等

C++深拷贝、浅拷贝

提问：

1. C++中：Cls是类名，Cls a = b会触发什么？



## 计算机系统工程导论

# ①命名设计考虑的因素

# 从抽象命名方案到命名实践

约束、需求与环境，是如何影响命名设计的？

命名是如何影响用户体验的？

命名是如何影响模块化的？

# 从抽象命名方案到命名实践

## 1. 讨论命名设计考虑的因素

### 1. 名称冲突

- 为什么会有名称冲突？
  - 模块间共享：用名称共享子模块，但模块使用不同命名上下文
  - 共享的收益：灵活性
  - 带来的问题：不同模块中可能包含相同名称的组件

名称冲突！

### 2. 元数据 (Metadata)

- 元数据：
  - 对象重要信息，但不在对象内部或在对象内部却难以查找
  - 名称、上下文等，都属于对象的元数据
  - 文件：
    - 名称、唯一标识符、类型、时间、版本、所有者、witness等

回忆：UNIX文件系统是怎么做的？

### 3. 地址 (address)

- 名称重载的特殊类型：与物理位置相关的名称重载
  - 地址是物理位置，或可映射到物理位置 (parsing)
    - 例：寄存器、内存地址、扇区号、IO端口、IP地址……
  - 用作定位器 (locator) —— 优点1
- 物理位置的几何特性，通常会对应到连续自然数地址
  - 从而支持算术操作，并映射到几何位置
  - 带来查找的高效性 —— 优点2

### 4. 唯一名称 (Unique Name)

- 唯一名称：不会发生冲突的名称
- 什么样的名称害怕冲突？
  - 权限、资源、资产、标记
- 如何生成唯一名称？五种方法：
  - ① 连续整数 (防止溢出)
  - ② 时间戳 (不会溢出)
    - 例：[查]UNIX系统时间为什么从1970-01-01开始？
  - ③ 伪随机数发生器 (PNG)
    - 例：[查]什么叫碰撞 (collision)

唯一名称的其他用途：临时ID

### 5. 用户友好性

- 设计冲突1：
  - 用户友好的名称容易冲突、冗余
  - 不易冲突、高效的名称难以记忆
- 能否同时兼顾？

UNIX文件系统：  
在机器层面上使用inode号、块号  
在人机接口使用文件名、路径名  
二者之间加入间接层


### 6. 生命周期

- 名、值和绑定三者的生命周期
  - 通常情况：名和值的生命周期长于绑定
  - 例：
    - 姓名电话、指针内存单元、域名/IP
  - 名和值都在，解绑定时是否可能发生问题？

## 2. 实例 (URL) 与现实事例

# 1.名称冲突

- 为什么会有名称冲突？
  - 模块间共享：用名称共享子模块，但模块使用不同命名上下文
    - **共享的收益：灵活性**
    - **带来的问题：不同模块中可能包含相同名称的组件**



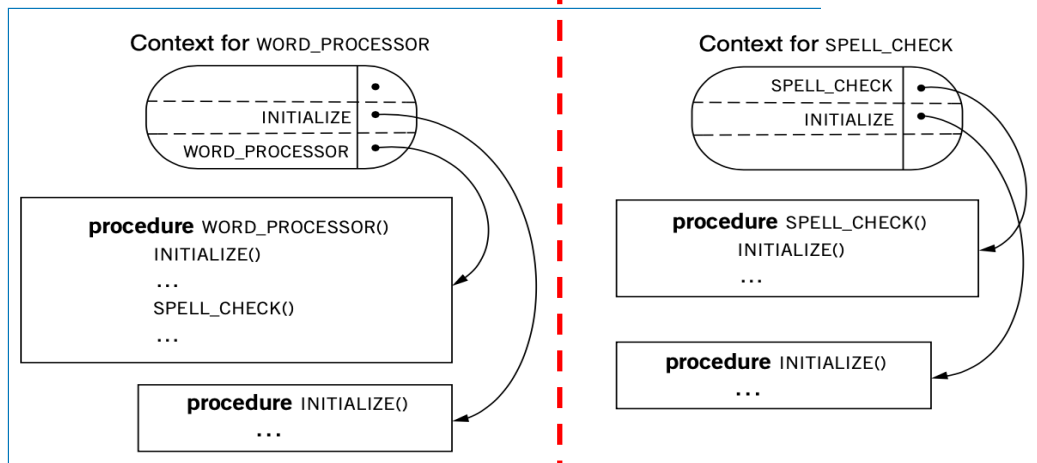
名称冲突!

# 1.名称冲突

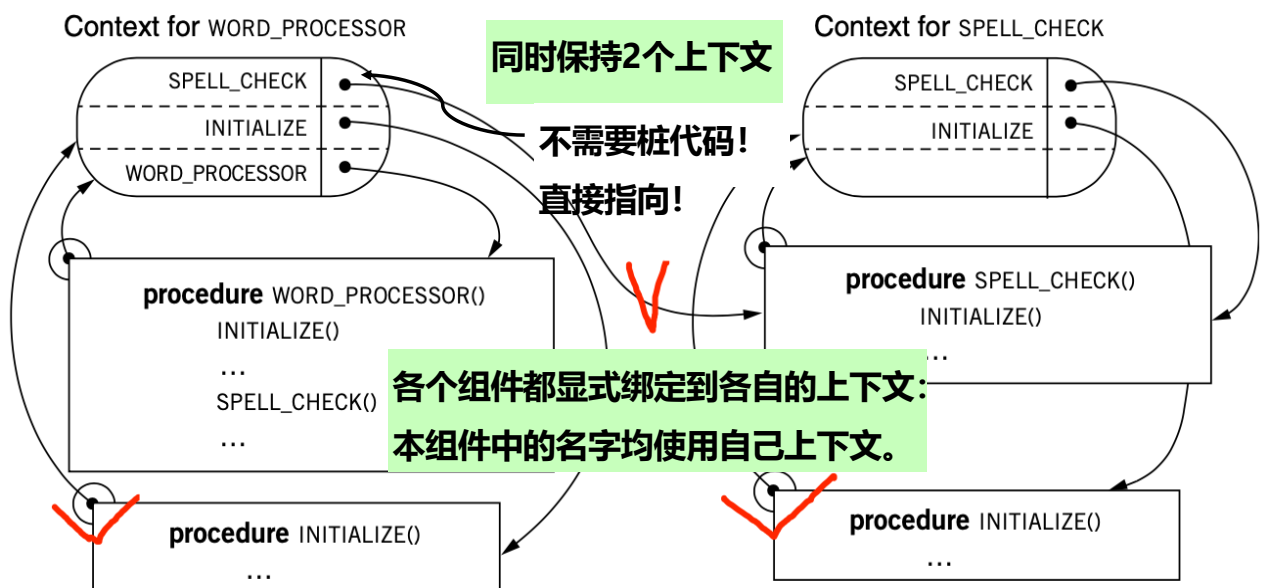
- 如何解决名称冲突？
  - 可以要求所有模块内部命名使用统一的命名方案吗？
  - 可以打破模块边界、互相交换模块内部的命名方案吗？
  - **破坏了模块化共享的思想！**
    - 只有抽象模块化才是计算机设计的目标，而非仅仅模块化。
    - 模块化抽象是我们做事的方式——Liscov
  - **思路：让各个模块在自己上下文中运行，间接交叉引用**

# 例

- A希望使用B的spell\_check, AB均有initialize函数
  - 如果合并上下文, 那么initialize函数不唯一
  - 如果不合并, 如何关联函数与上下文? ←—— 改名不是好办法!



## 方法1: 显式上下文

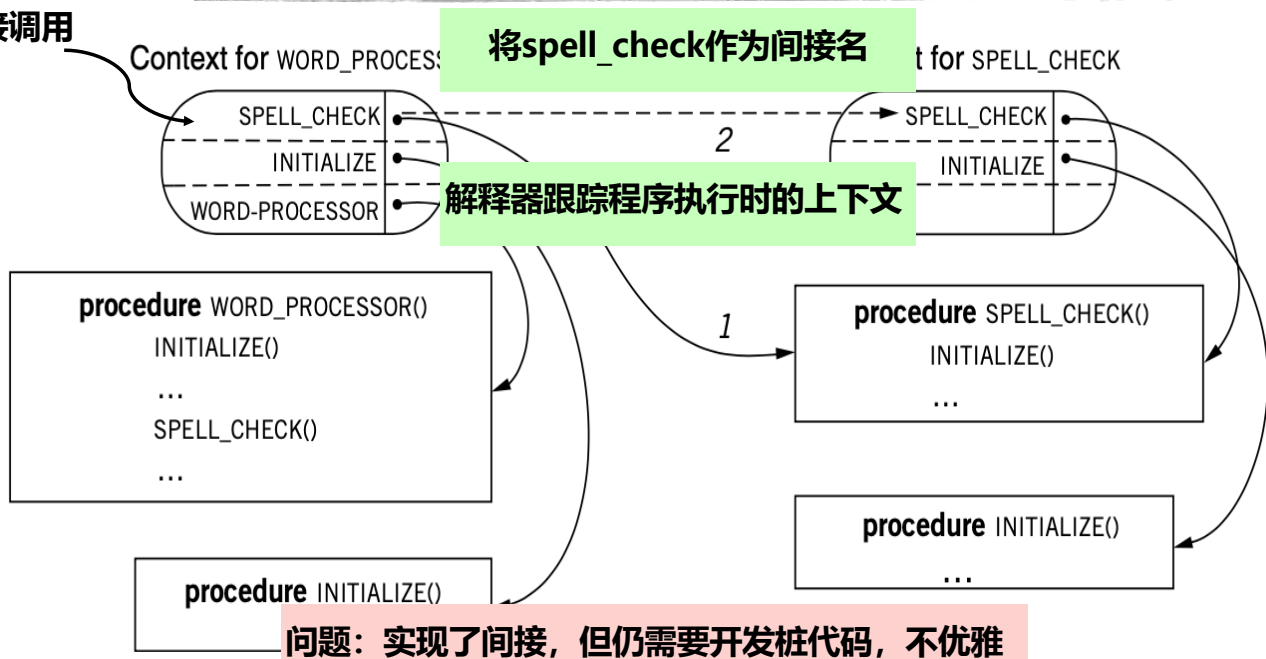


问题: 要修改组件, 修改的难度大、复杂度高

## 方法2：跟踪上下文

需要桩代码！

间接调用



## 更为优雅的做法：闭包

- 更优雅的closure(闭包)：打包自带静态上下文
  - 不同模块的initial函数能够找到自己的上下文，不仅仅靠名称引用
  - 语言近年来通常增补了该功能，在函数定义时使用结构来定义闭包

```
def make_averager():  
    series = []  
    def averager(new_value):  
        series.append(new_value)  
        total = sum(series)  
        return total/len(series)  
    return averager
```

closure →

free variable →

- C++11, Java, ...



## 现实中的例子：编译器处理名称冲突 符号修饰 (Name Decoration)

符号修饰：用改名处理冲突，正确、简单、高效。

### ● 名称冲突！

- 代码符号与已有的库中的符号冲突
- 函数重载出现同名函数
- 命名空间 (namespace) 合并后也会出现命名冲突

## 现实中的例子：编译器处理名称冲突 符号修饰 (Name Decoration)

符号修饰：用改名处理冲突，正确、简单、高效。

### ● 修饰方法：显性标识上下文

- 例如：
  1. UNIX下的C语言全局的变量和函数编译后符号名前加 “\_”，Fortran中的符号前后都加 “\_”。
  2. 局部变量：变量中加入函数名称（上下文）
  3. 重载：函数名加入函数特征

## 2.元数据 (Metadata)

- 元数据：
  - 对象重要信息，但不在对象内部或在对象内部却难以查找
  - 名称、上下文等，都属于对象的元数据
  - 文件：
    - 名称、唯一标识符、类型、时间、版本、所有者、witness等

回忆：UNIX文件系统是怎么做的？

## 元数据的保存

- 系统预设的元数据，称为系统元数据
  - 预设了保存空间和访问方法
  - 例如文件中的大小、权限、所有者.....
- 通常的系统，除名称外不支持自定义元数据
  - 例如文件类型、版本、作者.....
  - 如何保存自定义元数据？ → 名称重载

# 名称重载 (name overloading)

- 名称重载：名称中放入满足引用功能之外的各种元数据

- 例：扩展名，用地址做名称.....
- 下面的重载见过几个？

Name	Some of the things that overload this name
solutions.txt	solutions = file content; txt = file format
solutions.txt.backup 2	backup 2 = this is the second backup copy
businessplan 10-26-2007.doc	10-26-2007 = when file was created
executive summary v4	v4 = version number
image079.large.jpg	079 = where file fits in a sequence; large = image size
/disk-07/archives/Alice/	disk-07 = physical device that holds file; Alice = user id
OSX.10.5.2.dmg	OSX = program name; 10.5.2 = program version
IPCC_report_TR-4	IPCC = author; TR-4 = technical report series identifier
cse.pedantic.edu	cse = department name; pedantic = university name; edu = registrar name
ax539&ttiejh!90rrwl	no (apparent) overloading

## 名称重载

- 重载**收益**：可以抽取重载的元数据
  - 重载**风险**：可能破坏模块化和抽象
- 但名称重载会产生问题吗？
- → **脆弱名称** (fragile name)
    - 思考：路径是一种重载，指示了拓扑结构，一旦移动了还能找到吗？
      - 当文件从一个目录(盘)移动到另一个目录(盘)，以前重载的文件路径就不再能用，需要修改
      - 但名称的修改又会影响共享
    - www网址常因为这种原因而404！（在资源名中重载了查找路径）

404

根本问题：重载会导致名称附加意想不到的改变！影响访问和共享

# 纯名称 (pure name)

- 没有重载的名称，称为纯名称 (pure name)
  - 仅可：
    - COMPARE
    - RESOLVE
    - BIND
    - UNBIND
  - 健壮的代价：名称无法包含额外的元数据！

## 3.地址 (address)

- 名称重载的特殊类型：与物理位置相关的名称重载
  - 地址是物理位置，或可映射到物理位置 (parsing)
    - 例：寄存器、内存地址、扇区号、I/O端口、IP地址……
  - 用作**定位器** (locator) ←——— **优点1**
- 物理位置的几何特性，通常会对应到连续自然数地址
  - 从而支持**算术操作**，并映射到几何位置
  - 带来查找的**高效性** ←——— **优点2**

# 地址的脆弱性

- 问题
  - 地址是否也有脆弱性?
  - **地址的指称物不能移动! 因为地址的重载信息与位置相关!**
- 建议
  - 隐藏地址
  - [设计方法: 用间接解耦模块] (decouple modules with indirection)

地址设计原则: 永远不应暴露地址给用户!

## 当指称物移动, 地址需要改名时.....

- **四种方案:**
  1. 查找所有**引用**该地址的地方, 全部**替换**成新地址
  2. 附加**基于属性的查找**功能, 引用者可找到新地址
  3. 保留旧地址, 采用**间接层**, 随时**解析**到新地址
  4. 保留旧地址, 设置**转发器**进行**转发** (也是间接层)

根本方法: 遵守原则, 用间接层隐藏地址, 不使用地址做名称

# 使用纯名称可以防止名称脆弱性吗？

- 可以防止，但.....

## 4. 唯一名称 (Unique Name)

**唯一名称**：不会发生冲突的名称

- 什么样的名称害怕冲突？

- 权限、资源、资产、标记

唯一名称的其他用途：临时ID

- 如何生成唯一名称？**五种方法**：

- ① 连续整数（防止溢出）

- ② 时间戳（不会溢出）

- [查一查]UNIX系统时间为什么从1970-01-01开始？

- ③ 伪随机数发生器（PNG）

- [查一查]什么叫碰撞（collision）

## 4.唯一名称

### ④ 基于内容的唯一名称

- 内容就是唯一名称：长
- 更聪明的办法（短）：hash
  - 输出定长，输入任意，例如：sha256、sm3
  - [查一查]如何用hash碰撞攻击程序签名？

## 4.唯一名称

### ④ 基于内容的唯一名称

- 矛盾：
  - 重载的**收益**：可鉴别（witness）
  - 重载的**风险**：易碎性（fragility）
- **适用**：
  - 长期数据存储系统
  - 安全数据存储系统（检错、防篡改）

## 4.唯一名称

- ⑤ 层级命名方案
  - 层级的特性：代表（delegation）
    - 例：域名、MAC address

思考：如何选定计算机的唯一标识？（用作版权保护等）

并不简单。

## 5.用户友好性

- 设计冲突1：
  - 用户友好的名称容易冲突、冗余
  - 不易冲突、高效的名称难以记忆
  - 能否同时兼顾？

**UNIX文件系统：**

在机器层面上使用inode号、块号  
在人机接口使用文件名、路径名  
二者之间加入间接层



# 5.用户友好性

## ● 设计冲突2:

- 有的系统大小写敏感
- 有的系统大小写不敏感
- 如果跨系统复制文件怎么办?
  - 补救措施, 以DNS、Mac文件系统为例:
    - case-coercing (大小写强制转换): 强制转换全部大写或小写
    - case-preserving (大小写保持): 大小写仅用作显示用途

历史遗留错误 (违反最小惊讶原则)

# 5.用户友好性

## ● 更友好的方式

- icon
- 超链接与交叉链接系统
  - 交叉链接系统 (cross-linking): 谁引用了我?
    - "Alas, poor Yorick! I knew him, Horatio" (Hamlet)

## 6.生命周期

---

- 名、值和绑定三者的生命周期
  - 通常情况：名和值的生命周期长于绑定
    - 例：
      - 姓名/电话，指针/内存单元，域名/IP
    - 名和值都在，解绑定时是否可能发生问题？

## 6.生命周期

---

- 名称带来的问题：悬垂引用 (dangling reference)
  - C程序中的悬垂指针、UAF、Double Free
  - 应对：动态检查
- 值带来的问题：内存垃圾 (orphan/lost object)
  - C程序中的泄露 (storage leak)
  - 应对：内存垃圾收集 (garbage collection, GC)
  - 回顾：UNIX文件系统的引用计数

# 展望

- 名称是系统的基本构件，后面的章节：
  - C/S架构的硬模块化中，用名称来定位服务
  - 计算机虚拟化中，虚拟内存是地址命名系统
  - 〈性能：缓存是renaming 设备〉
  - 〈原子性：事务使用临时名称和最终绑定〉
  - 〈安全〉
    - principal - 代表被管理的主体
    - key - 难以猜测的名称
    - name-to-key 绑定

## ②URL实例与现实事例

# URL是什么

- URL是什么？统一资源定位符（Uniform Resource Locator）
  - 用以标记数字资源
  - 例
    - <http://web.ouc.edu.cn/April/www/home.html>

## 1.URL的名称发现方法

- 上层：从哪来
  - ① URL的初始来源来自于浏览器内置、文档、其他媒体
  - ② 从URL得到URL：超链接
    - URL以超链接（hyperlink）形式出现在超文本（hypertext）中
    - 例：<a href="http://web.ouc.edu.cn/April/www/home.html">April's page</a>
      - <a href="">txt</a>表示链接类型的标签，其中
        - a，超链接标签
        - href属性，表示超链接(h)引用(ref)的URL
        - txt是显示的文本

## 2.URL的解析

- 下层：何处去

- ①绝对URL

- 例：http://web.ouc.edu.cn/April/www/home.html
      - http：协议类型，用于定位解析器
      - web.ouc.edu.cn：服务器域名，用于向DNS获得IP
      - April/www/home.html：路径名，用于服务器查找文件

## 2.URL的解析

- 下层：何处去

- ②相对URL

- 例：<a href="about.html">关于</a>
      - URL使用当前所在网页的URL的默认上下文
      - 除非“base elemen” t存在 → 覆盖（override）默认上下文
        - <base href="http://web.ouc.edu.cn/April/">

# \*\*误区与安全攻击

- 误区：URL路径语义与文件系统不同，URL勿与文件系统混淆
  - 符号链接导致路径不唯一
  - URL路径与文件系统路径——映射存在风险
  - 例：“..”在URL和文件系统中有不同的含义
- ../ 攻击 (Dot dot slash attack)
  - 目录遍历攻击
  - 安全类课程深入探讨

## 3.大小写敏感的问题：不同处理

- 域名
  - 大小写不敏感
- 协议名
  - 依赖于浏览器实现
  - [试一试] Firefox、Safari、Chrome、Edge
- 域名后的部分
  - 依赖于服务器协议实现
  - 有些服务器将路径用于文件，又依赖于文件系统
  - [试一试] 新浪、QQ.com.....

## 4.易产生的误用

- 如果同一个对象的链接有两种形式：

1. <http://web.ouc.edu.cn/april/home.html>

2. <http://web.ouc.edu.cn/april/www/home.html>

那么网页中<a href = "contacts.html"> 指向哪里？

取决于\_\_\_\_\_的解释？

## 5.名称重载与脆弱性

- http协议无状态，有的应用借助URL重载状态。例：
  - https://www.amazon.com/dp/0670672262/ref=redir\_mobile\_desktop?\_encoding=UTF8&%2AVersion%2A=1&%2Aentries%2A=0
  - 将状态放入URL的路径，传给服务器
  - 相当于增加了一层状态协议（session layer）
- 有的开发者用URL包含服务器名称和文件系统信息。例：
  - http://HPserver14.pedantic.edu/disk05/science/geophysics/quakes.html
  - 问题：脆弱性。解决方法：间接URL跳转：  
http://quake.org/library/quakes.html
- [查一查]希望取代URL的更持久化的命名方案：PURL、URN、DOI

革命尚未成功

## \*\*DOI: 数字对象唯一标识符 Digital Object Unique Identifier

- DOI: 一套识别数字资源的机制, 如视频、报告或书籍等等。目前国外大型出版商大多使用DOI对数字资源进行标识。
  - 特点: 唯一性、持久性、兼容性、互操作性、动态更新。
  - DOI码由前缀和后缀组成, 之间用 “/” 分开。前缀由IDF确定, 以 “.” 分为两部分: 10.登记机构代码。后缀由资源发布者自行指定。
  - 例:
    - ▷ DOI: [10.1109/Trustcom/BigDataSE/ICCESS.2017.242](https://doi.org/10.1109/Trustcom/BigDataSE/ICCESS.2017.242)

## 现实事例 #1: 碰撞抹去笑脸

- 某高校的选课系统, 老师可看到班级中每位同学上传的照片
- 某学期, 出现了大量同一名同学的照片
  - 提示, 该同学名字叫smiley
- 原因: smiley.jpg





# 现实事例 #2：脆弱的名称

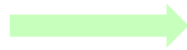
- 公司（学校）合并
  - 导致域名合并，为什么要合并？
    - 因为域名重载了商标等信息
  - 产生的问题：电子邮件地址碰撞
  - 讨论：
    - 哪些方案可用？考虑成本、灵活性、人机工程等
    - 间接能给我们什么样的启发？

# 现实事例 #3：更脆弱的名称

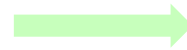
- 邮政编码
  - 1位国家区域. 2位片区. 2位站点
  - 非连续的，仅用于自动分拣、装箱运输
  - 因为人口增多，021拆分为021和024
    - 美运信用卡（American Express）的系统中，邮编改变等同地址移动，影响信用评分和验证。所以其系统内仍使用旧邮编
    - 但是，有些公司拒绝账单地址与邮寄地址不同的下单，导致冲突
    - 如何解决？

# 总结

复习与回顾



命名实践问题



实例



## 重要术语中英文对照



命名冲突: name collision

闭包: closure

名称修饰: name decoration

元数据: metadata

安全校验: witness

名称重载: name overloading

地址: address

碰撞: collision

哈希: hash

大小写敏感: case-sensitive



## 重要术语中英文对照



大小写保持: case-preserving  
大小写转换: case-coercing  
存储 (内存) 泄露: storage (memory) leak  
(内存) 垃圾收集: garbage collection  
客户端、服务器、协议: client, server, protocol  
超链接: hyperlink  
超文本: hypertext  
统一资源定位地址: URL  
超文本标记语言: html  
域名系统: DNS



## 文献阅读与作业



- 阅读参考书的4.4: "Case study: The Internet Domain Name System (DNS)".
  - 读完后如果不确定自己是否基本理解了DNS的原理, 可以用讲义上的自测题进行自测 (见讲义)
  - 自测题如果感觉有困难, 那么可与同学或助教讨论并再次阅读, 并思考
    1. “recursive query”是什么? 能带来什么收益?
    2. DNS 的 “hierarchical design”是什么样的? 能带来什么收益?
    3. DNS的设计有哪些缺点?



# 文献阅读与作业



- 阅读参考书的4.4: "Case study: The Internet Domain Name System (DNS)".
  - 问题（作为作业上交）：
    - 1.DNS的设计目的是什么？
    - 2.DNS是怎么运行的？
    - 3.为什么DNS设计成这样一种运行的模式？
  - 要求
    - 一般不超过一页A4纸，使用自己的语言书写
    - 没有唯一或绝对正确的答案，答案与考虑问题的场景或假设有关
    - 禁止复制他人文字和使用ChatGPT等工具生成
    - 7天内提交到Bb系统，晚交分数减半，晚交7天则无成绩



# 本章主要参考文献



- Jack B. Dennis. Segmentation and the design of multiprogrammed computer systems. Journal of the ACM 12, 4 (October 1965), 589–602.
- Robert S. Fabry. Capability-based addressing. Communications of the ACM 17, 7 ( July 1974), 403–412.
- Paul J. Leach, Bernard L. Stumpf, James A. Hamilton, and Paul H. Levine. UIDs as internal names in a distributed file system. In ACM SIGACT–SIGOPS Symposium on Principles of Distributed Computing, Ottawa, Ontario (August 18–20, 1982), 34–41.

# 第3章 结束

