vmware | SPRING & CLOUD FOUNDRY
开发者大会

分解应用程序提升
可部署性和可扩展性

Chris Richardson,

《POJOs in Action》作者，原 CloudFoundry.com 创始人

演讲目的

分解应用程序

如何提高可部署性与可扩展性

以及

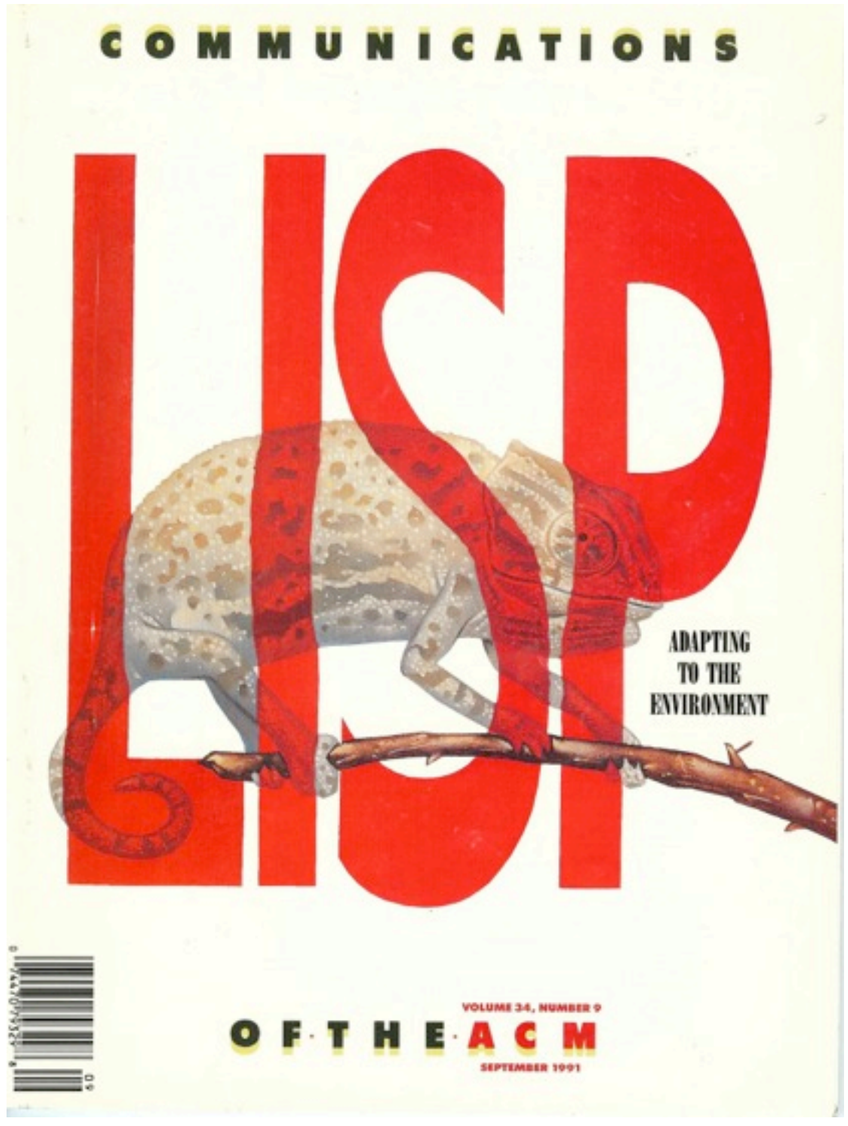
Cloud Foundry 有何帮助

Chris 介绍

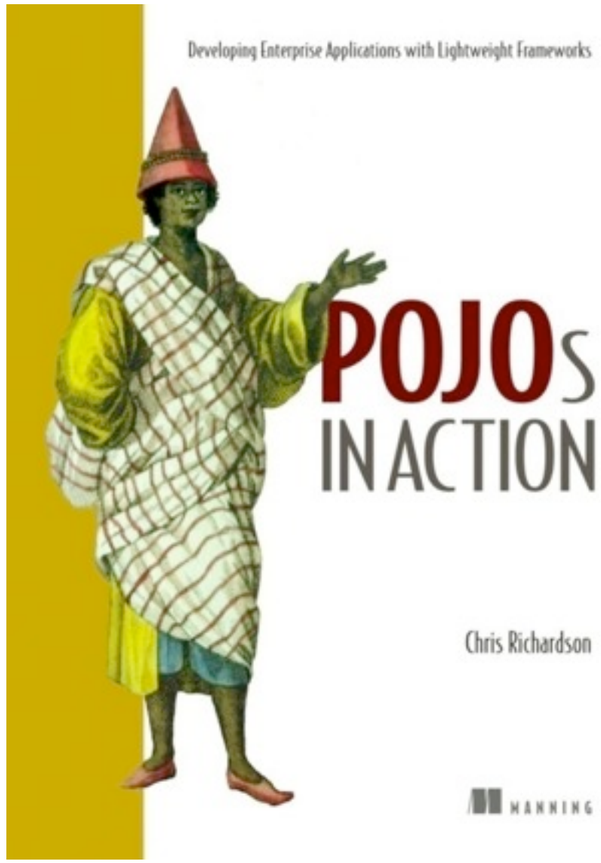

Aberdeen
Dundee
Glasgow
Londonderry / Derry
Belfast
United Kingdom
Hartlepool
Isle of Man
Leeds
Preston
Hull
Dublin
Ireland
Manchester
Leicester
Norwich
Birmingham
Ipswich
Northampton
Swansea
Basildon
Barnet
Bristol
A
Exeter
Portsmouth
Eastbourne
Plymouth


Medford
Klamath Falls
Yreka
Eureka
Redding
Elko
Chico
Reno
Nevada
Yuba City
Carson City
A
Sacramento
Santa Rosa
San Francisco
Stockton
San Jose
California
Fresno
Visalia
Las Vegas
Santa Maria
Bakersfield
Lompoc
Palmdale
Santa Clarita
Riverside
Hemet
Los Angeles
Oceanside
Yuma
Tijuana
Mexicali
Ensenada
Puerto Peñasco

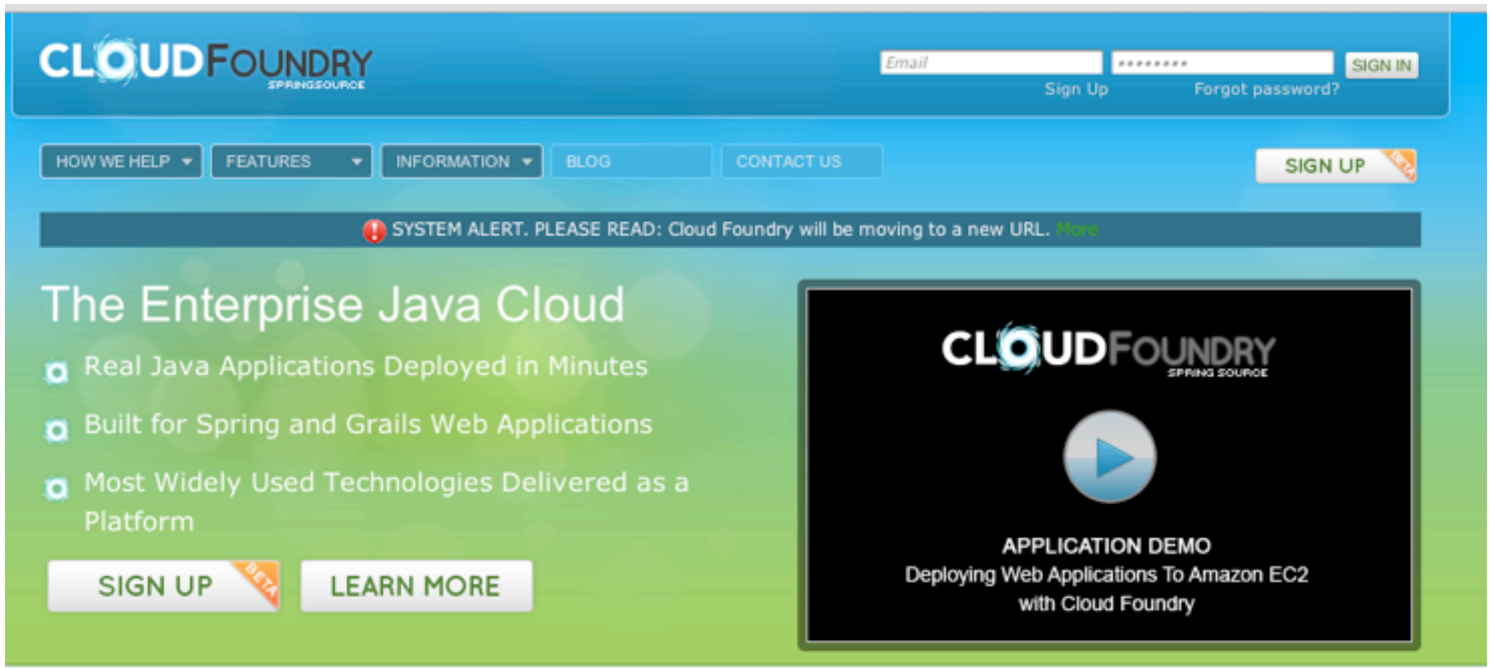
(Chris 介绍)


COMMUNICATIONS
LISP
ADAPTING
TO THE
ENVIRONMENT
VOLUME 34, NUMBER 9
OF THE ACM
SEPTEMBER 1991

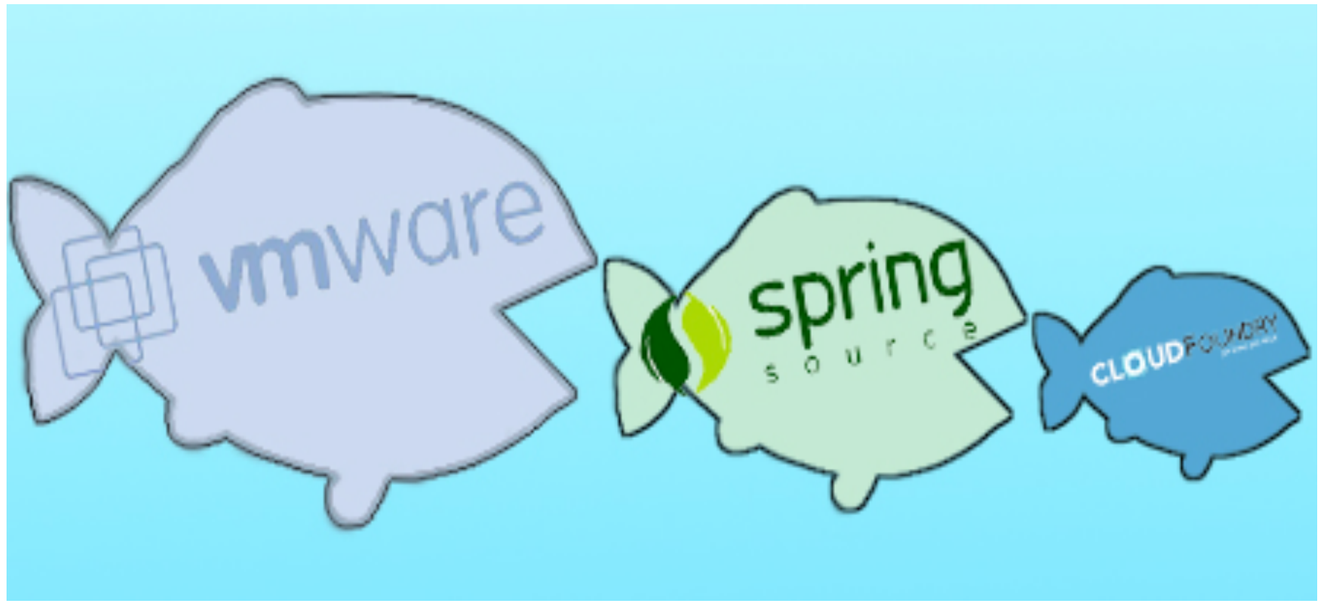
Chris 介绍


Developing Enterprise Applications with Lightweight Frameworks
POJOs
IN ACTION
Chris Richardson
MANNING

Chris 介绍


CLOUDFOUNDRY
SPRINGSOURCE
Email
SIGN IN
Sign Up
Forgot password?
HOW WE HELP
FEATURES
INFORMATION
BLOG
CONTACT US
SIGN UP
SYSTEM ALERT. PLEASE READ: Cloud Foundry will be moving to a new URL. More
The Enterprise Java Cloud
Real Java Applications Deployed in Minutes
Built for Spring and Grails Web Applications
Most Widely Used Technologies Delivered as a Platform
SIGN UP
LEARN MORE
CLOUDFOUNDRY
SPRING SOURCE
APPLICATION DEMO
Deploying Web Applications To Amazon EC2
with Cloud Foundry

Chris 介绍

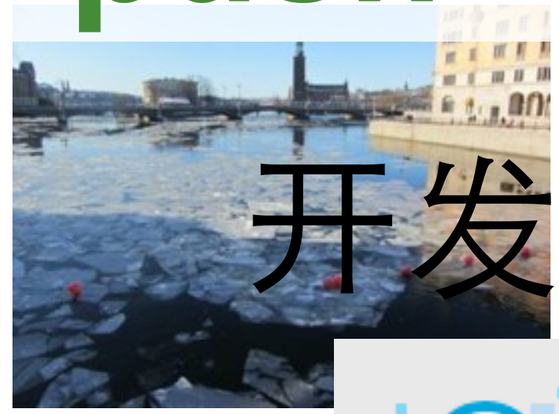
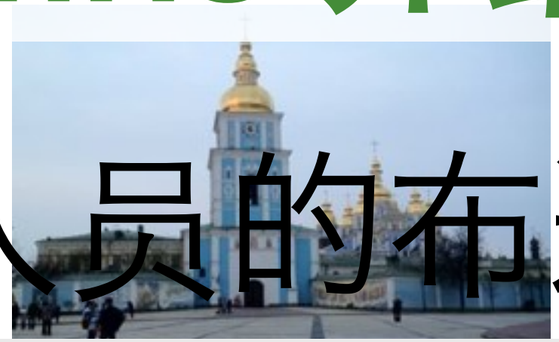


http://www.theregister.co.uk/2009/08/19/springsource_cloud_foundry/

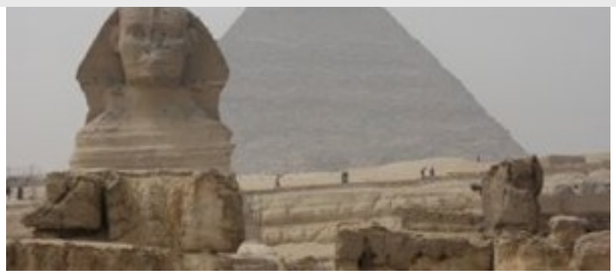
vmc push Chris 介绍

开发人员的布道师

vmc push Chris 介绍


开发人员的布道师



vmc push Chris 介绍

议题

- 整体性（有时不如人意）
- 将应用程序分解为服务
- 服务如何通信？
- 表示层设计
- Cloud Foundry 有何帮助

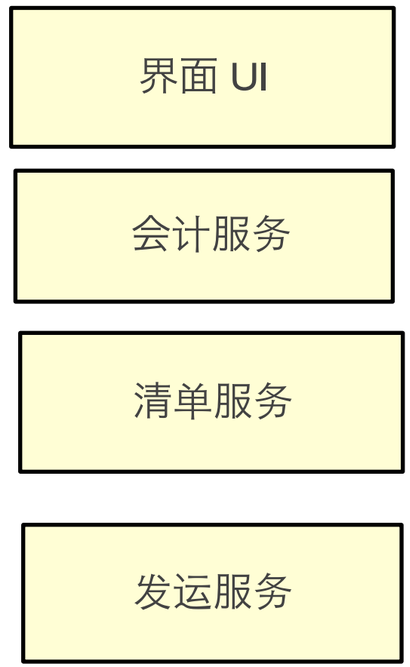
想象一下，您正在构建

电子商务应用程序

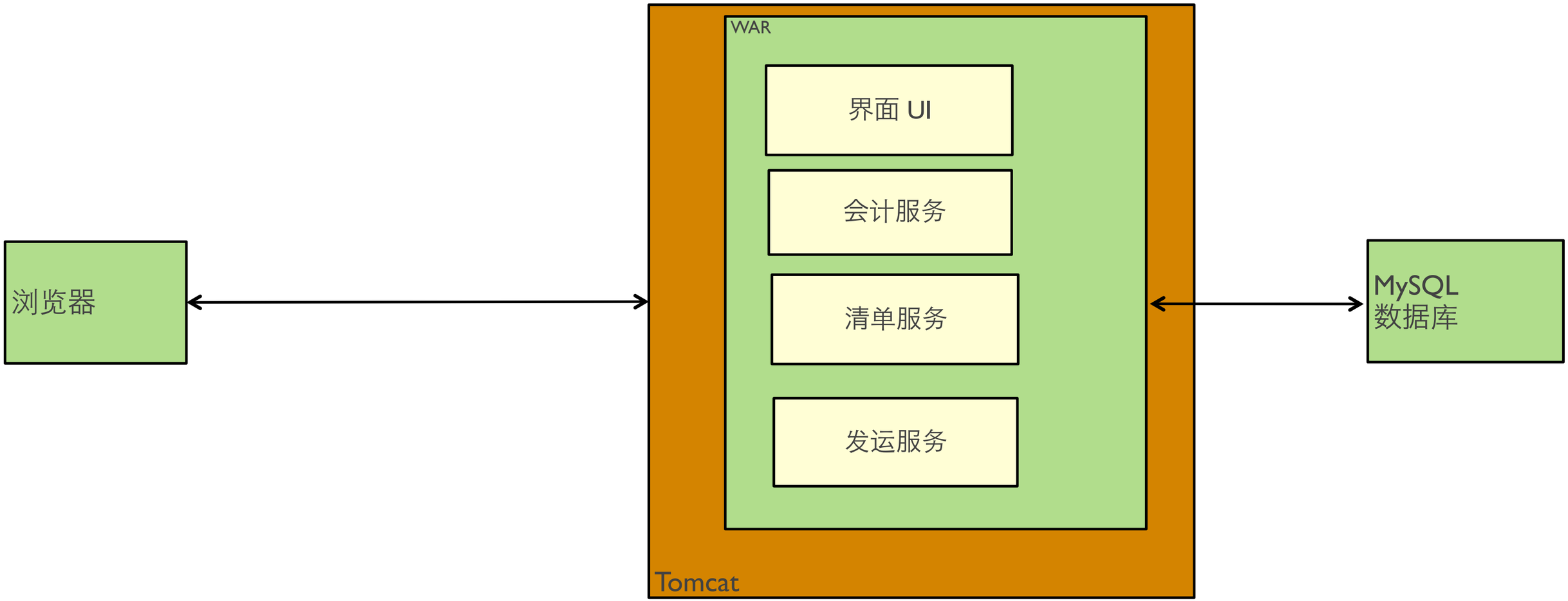
传统 Web 应用程序架构

传统 Web 应用程序架构

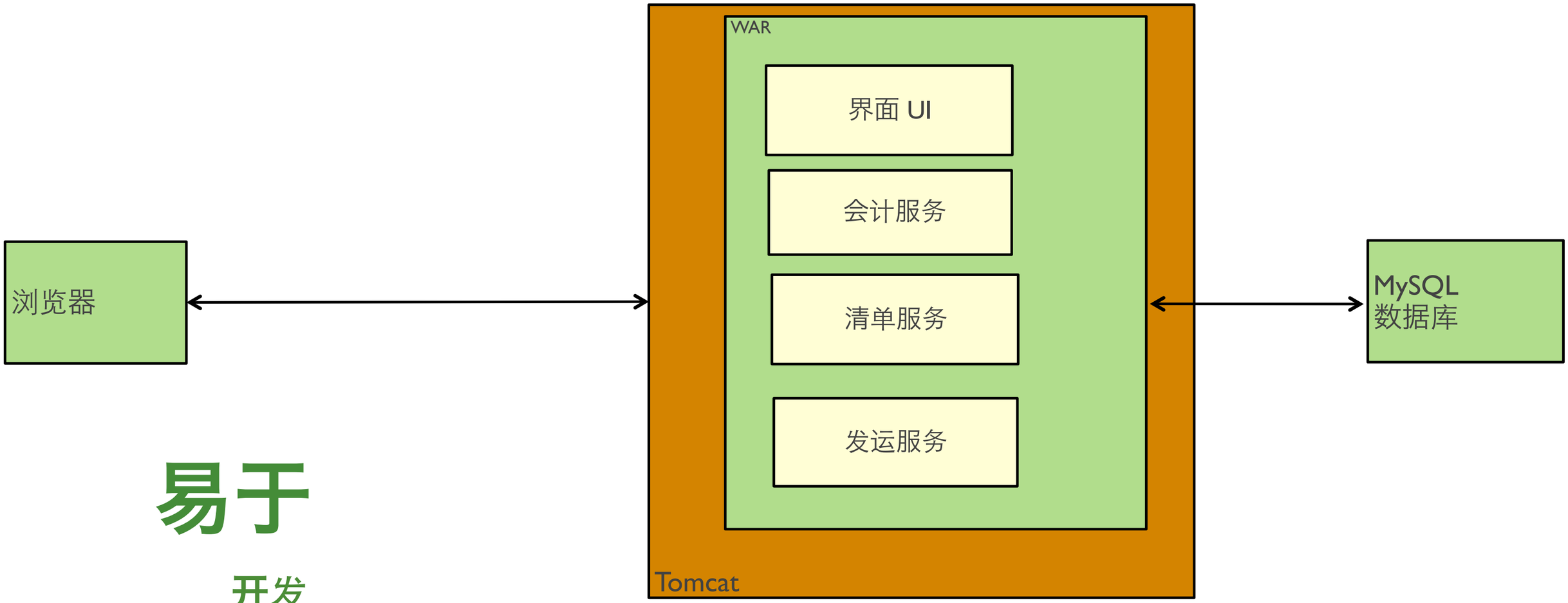
传统 Web 应用程序架构



传统 Web 应用程序架构



传统 Web 应用程序架构



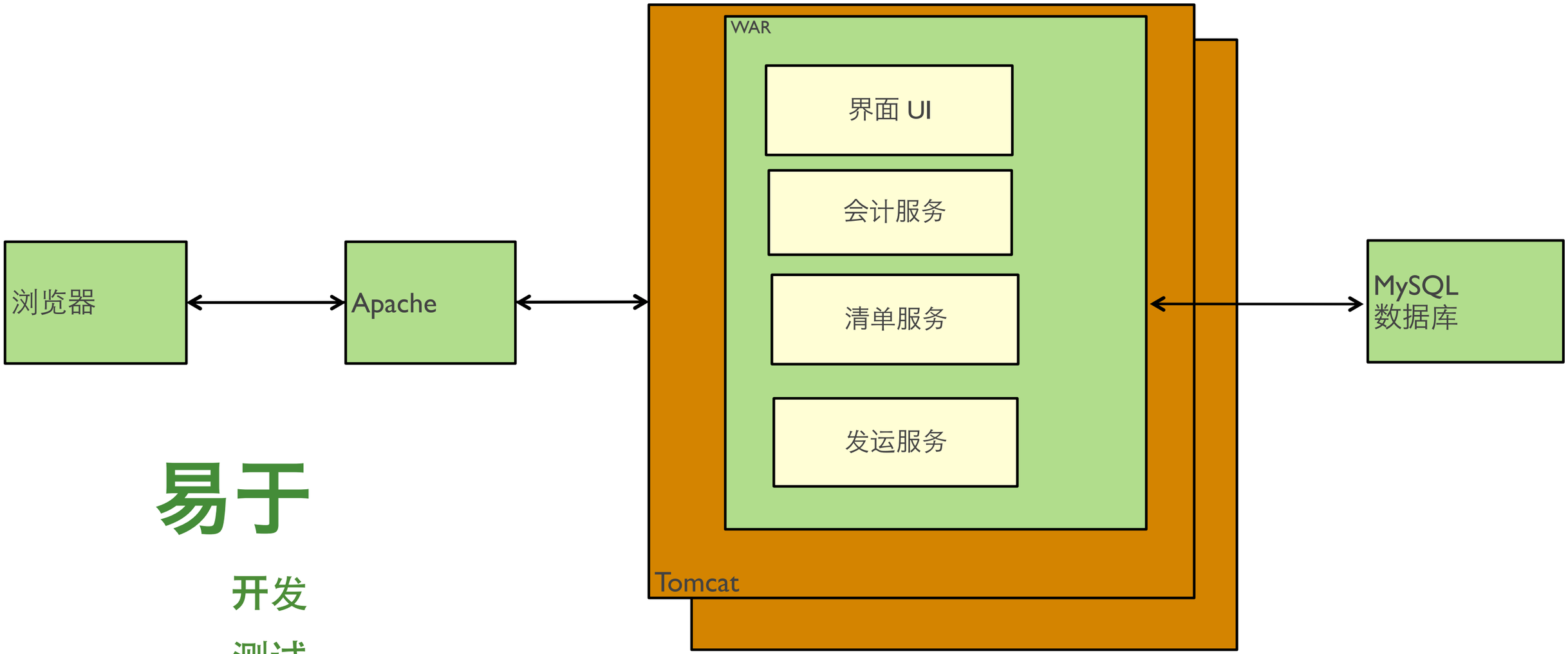
易于

开发

测试

部署

传统 Web 应用程序架构



易于

开发

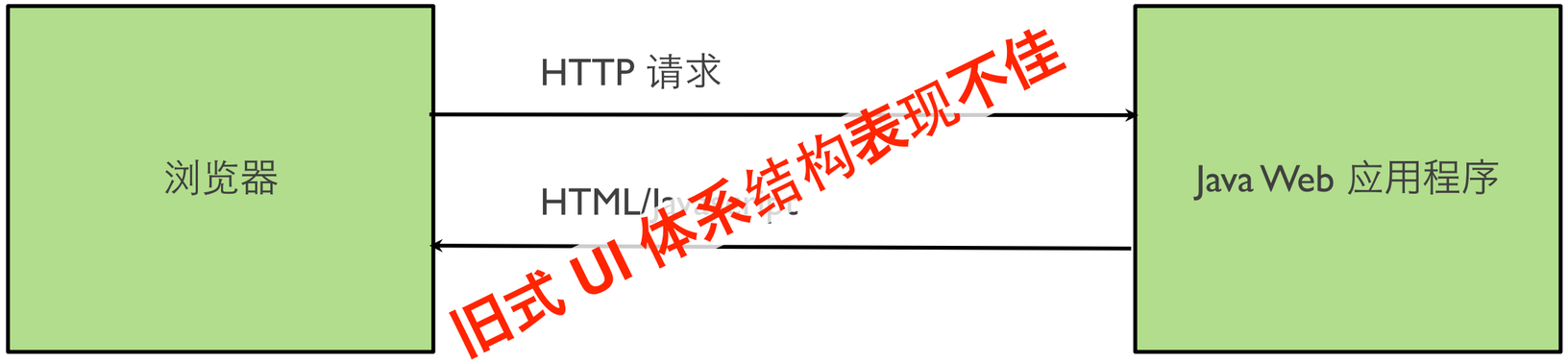
测试

部署

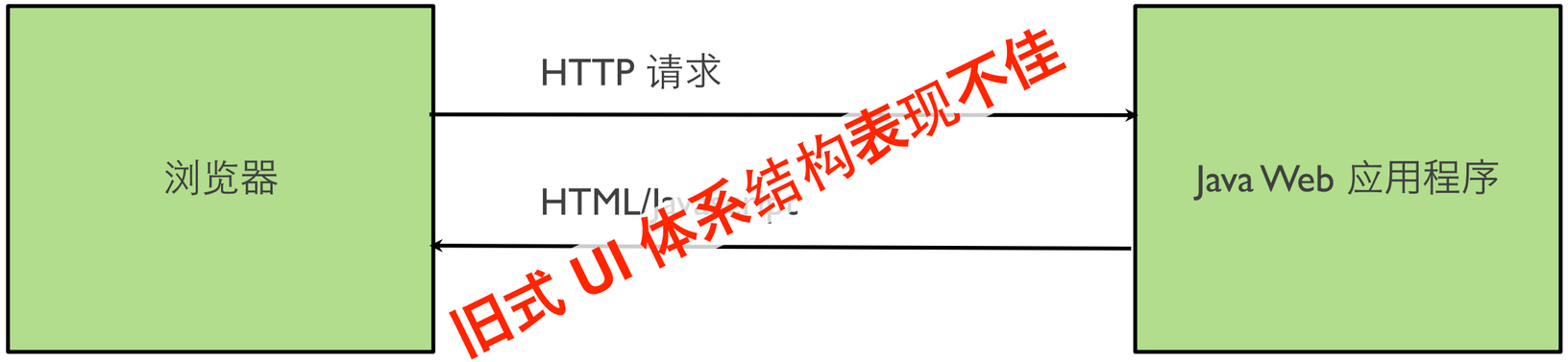
扩展

但是整体式体系结构
存在一些问题

用户期望丰富、动态



用户期望丰富、动态



实时 web ≅ NodeJS

令开发人员望而却步

频繁部署的障碍

- 变更一个组件需要全部重新部署
- 中断长时间运行的后台（如 Quartz）作业
- 增加故障风险

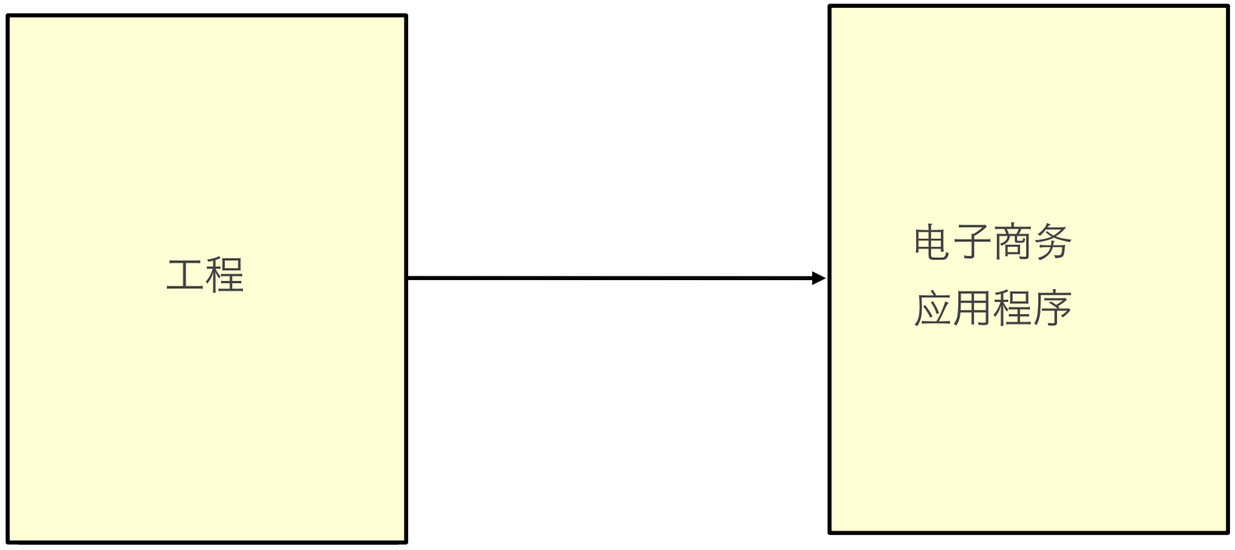
害怕变更

- 更新次数减少
- 例如，令 A/B 测试 UI 异常困难

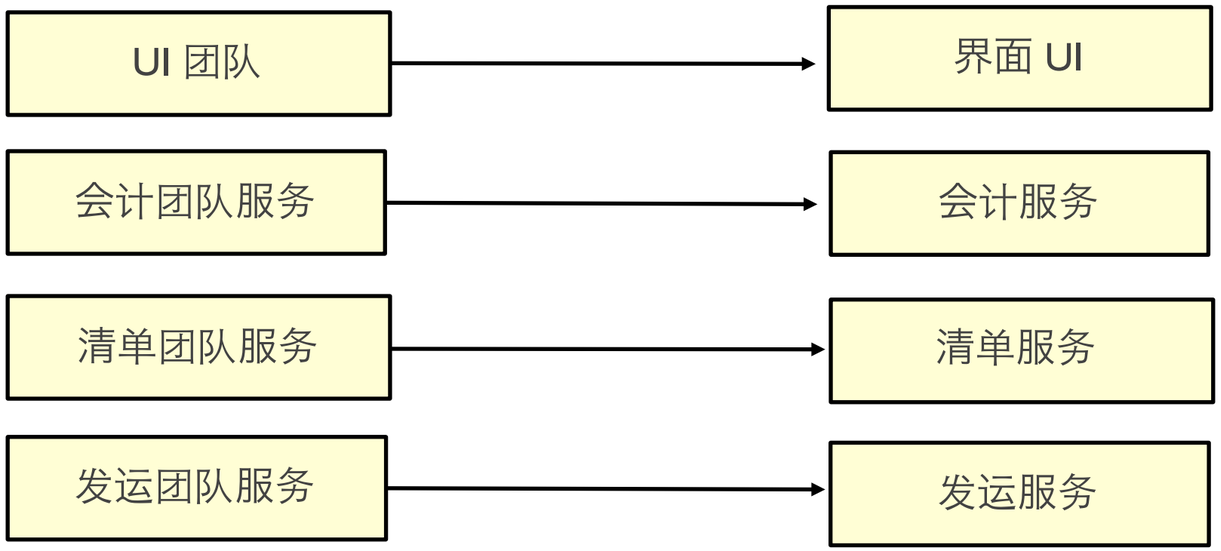
使 IDE 和容器过载

减缓开发速度

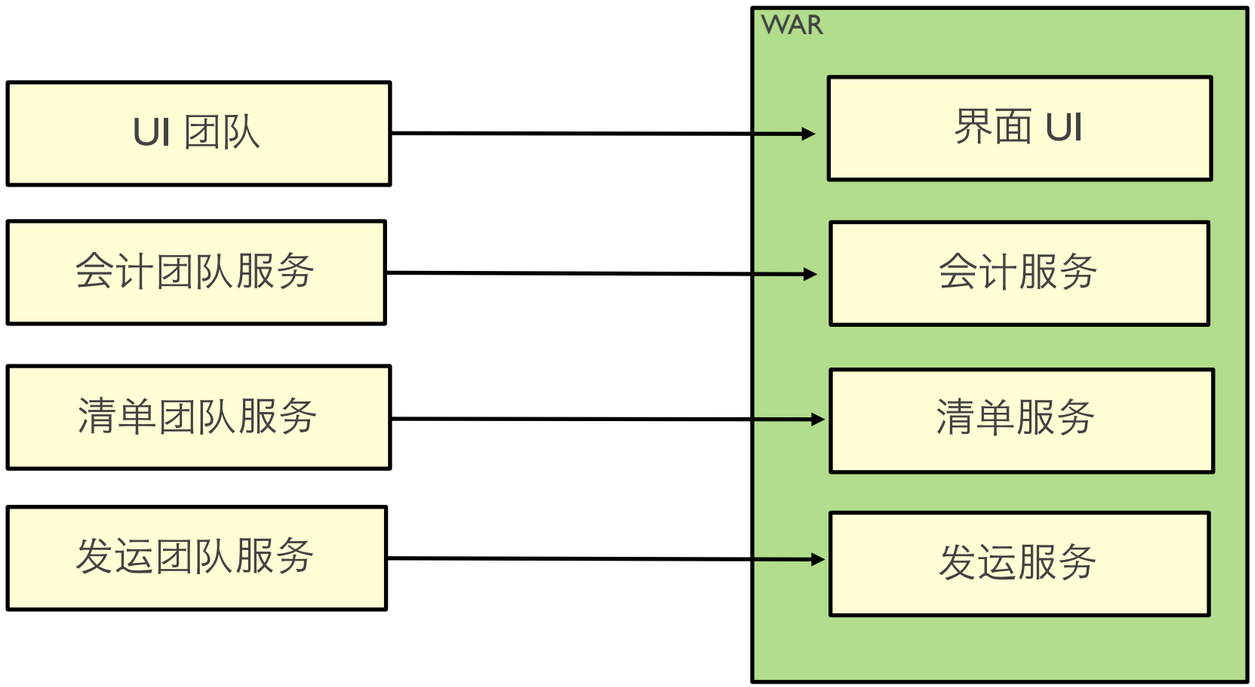
扩展开发的障碍



扩展开发的障碍



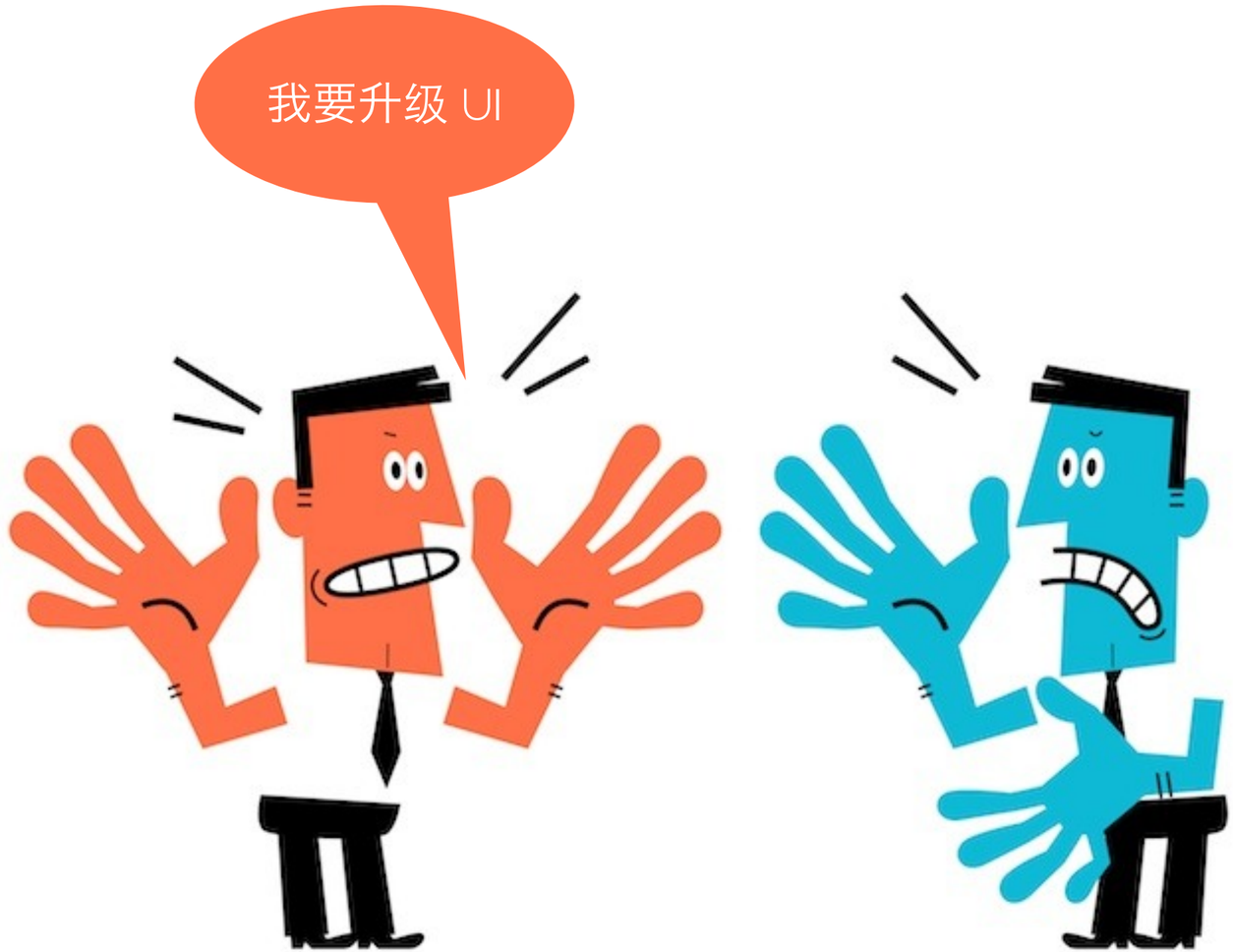
扩展开发的障碍



扩展开发的障碍

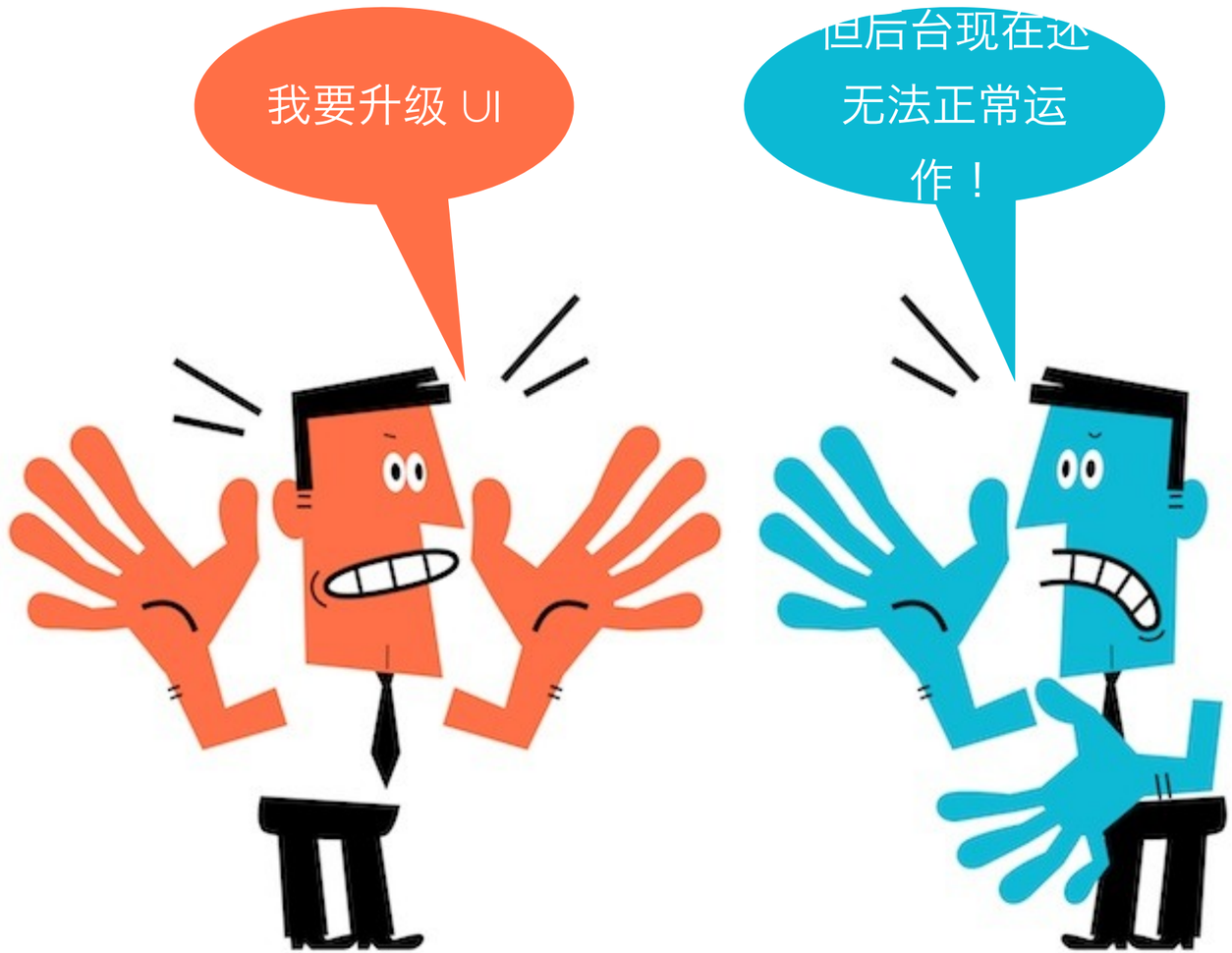
需要大量协调和沟通工作

扩展开发的障碍



需要大量协调和沟通工作

扩展开发的障碍

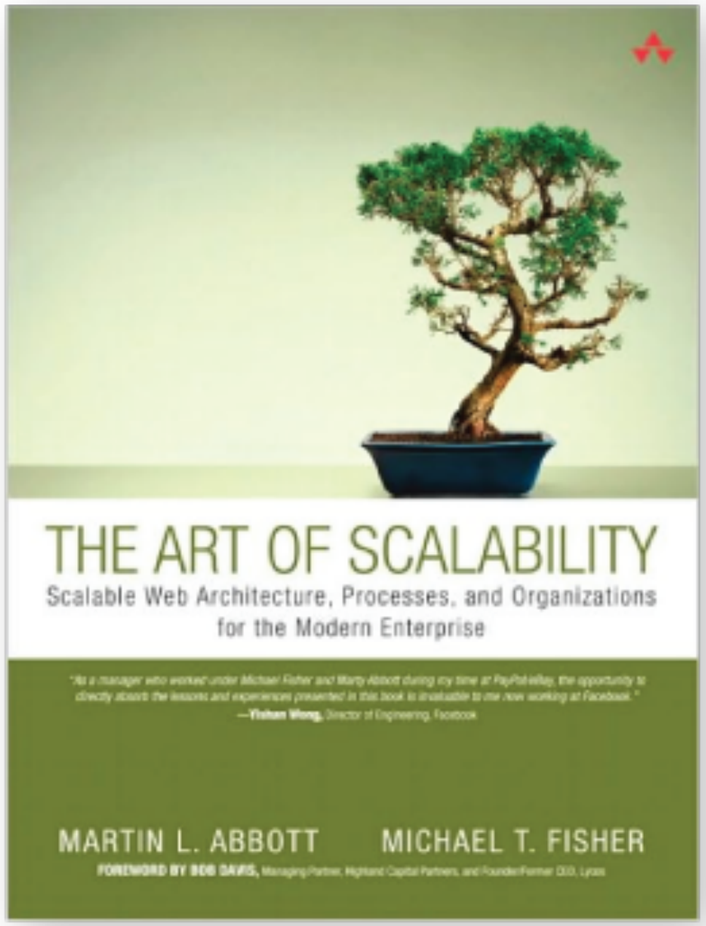


需要大量协调和沟通工作

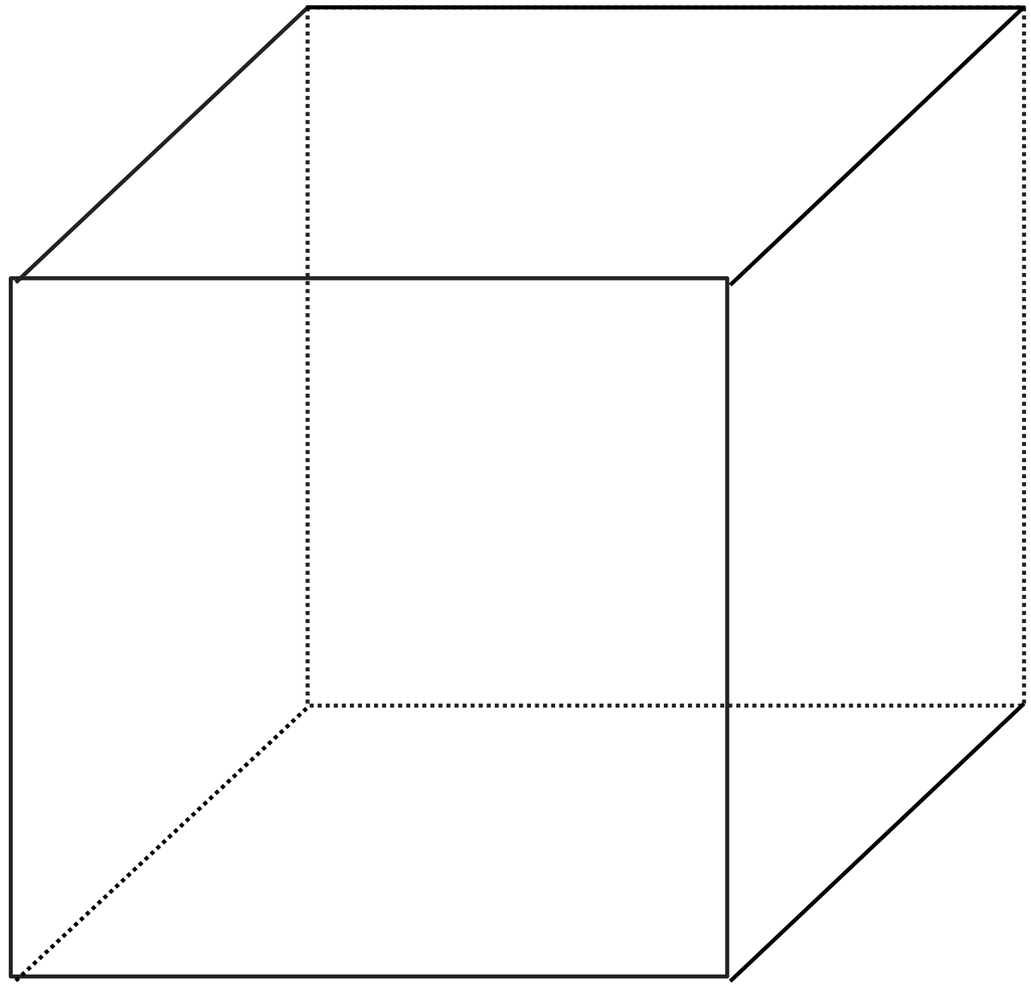
需要长期依赖于单一技术架构

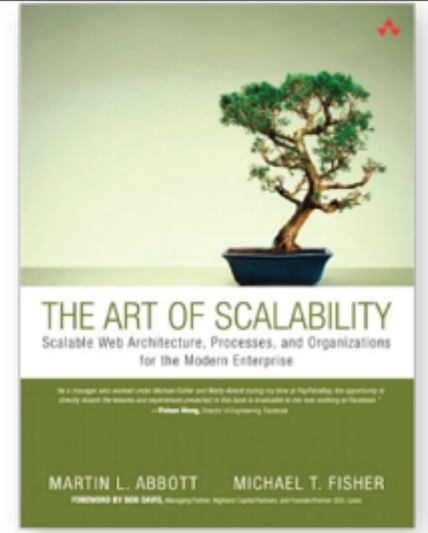
议题

- 整体性（有时不如人意）
- 将应用程序分解为服务
- 服务如何通信？
- 表示层设计
- Cloud Foundry 有何帮助

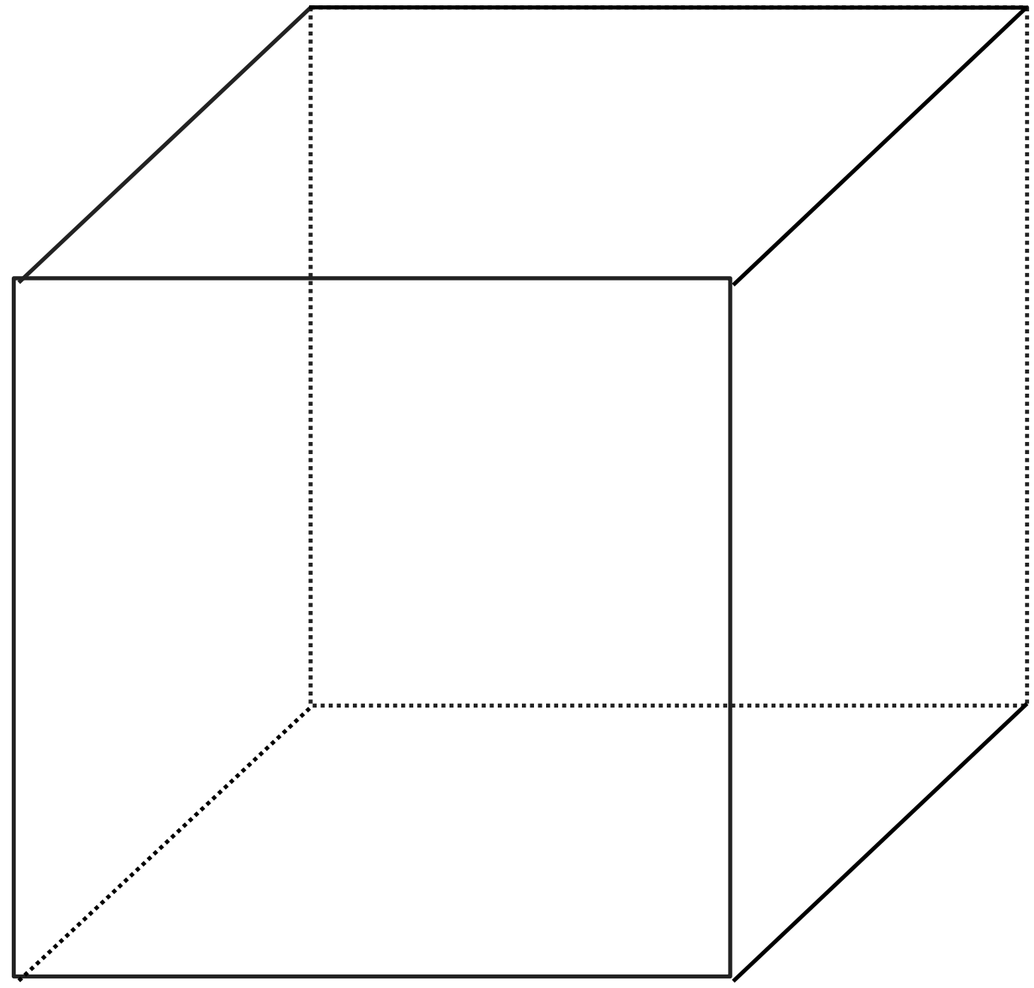

THE ART OF SCALABILITY
Scalable Web Architecture, Processes, and Organizations
for the Modern Enterprise
MARTIN L. ABBOTT
MICHAEL T. FISHER

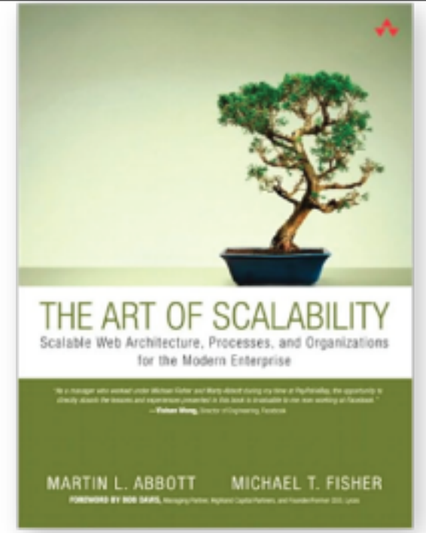
扩展立方体




THE ART OF SCALABILITY
Scalable Web Architecture, Processes, and Organizations for the Modern Enterprise
MARTIN L. ABBOTT
MICHAEL T. FISHER

扩展立方体

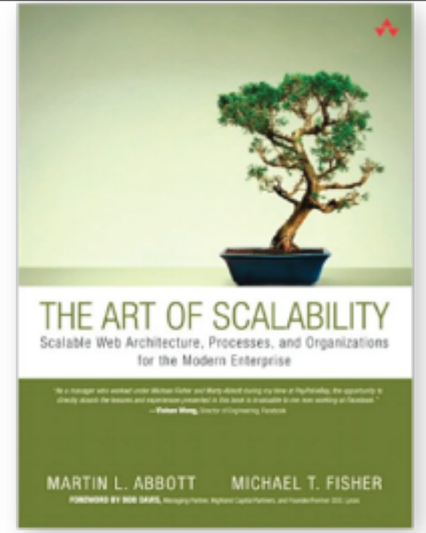


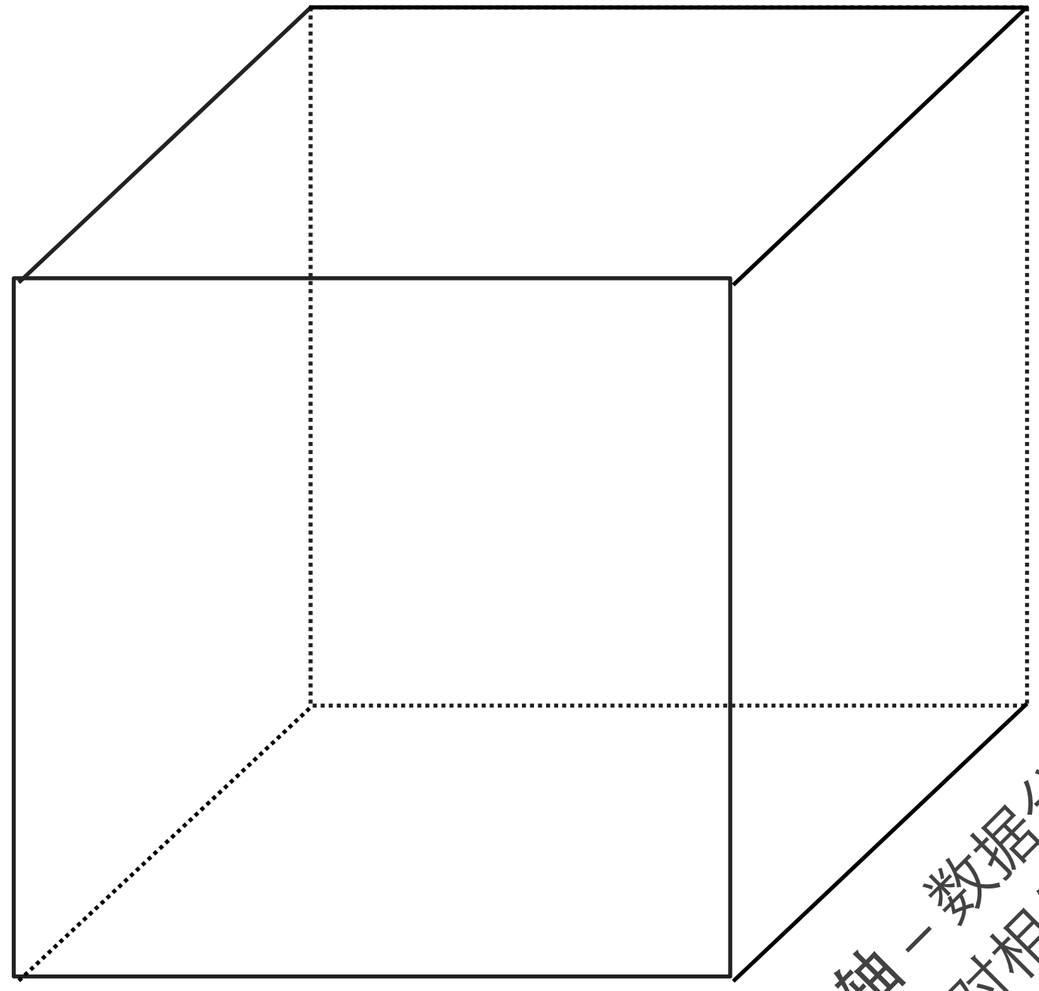


X 轴

- 水平复制

扩展立方体

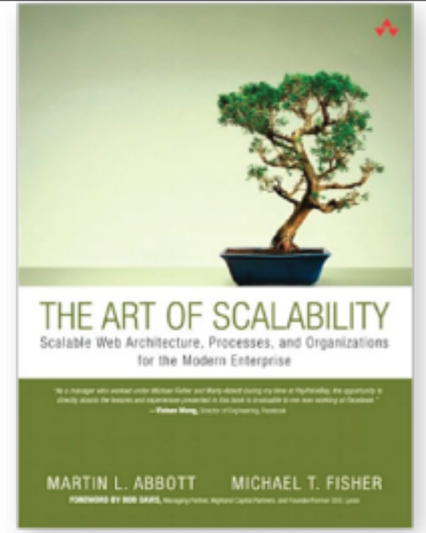




X 轴
- 水平复制

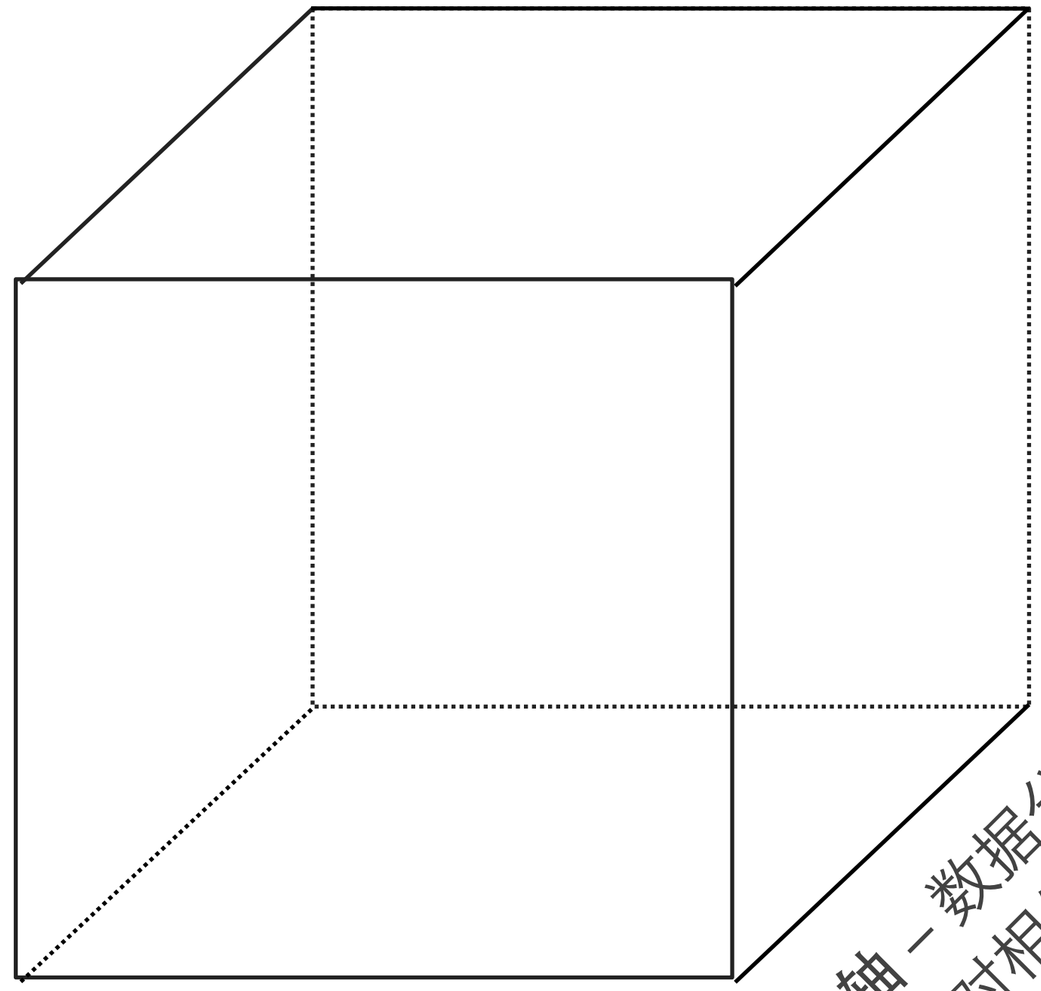
Z 轴 – 数据分区
通过对相似事务
进行分割来扩展

扩展立方体



Y 轴 -
功能分解

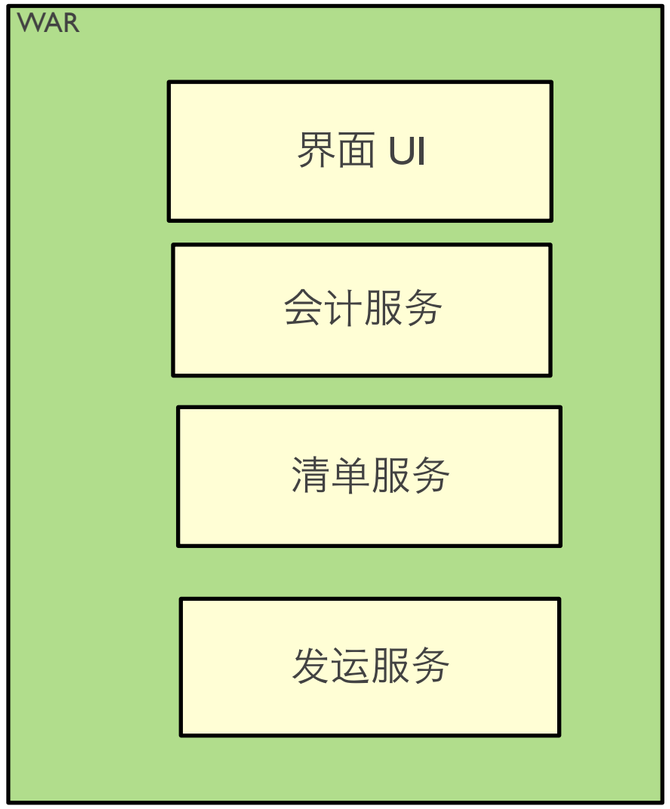
通过对不同事务进行分割来扩展



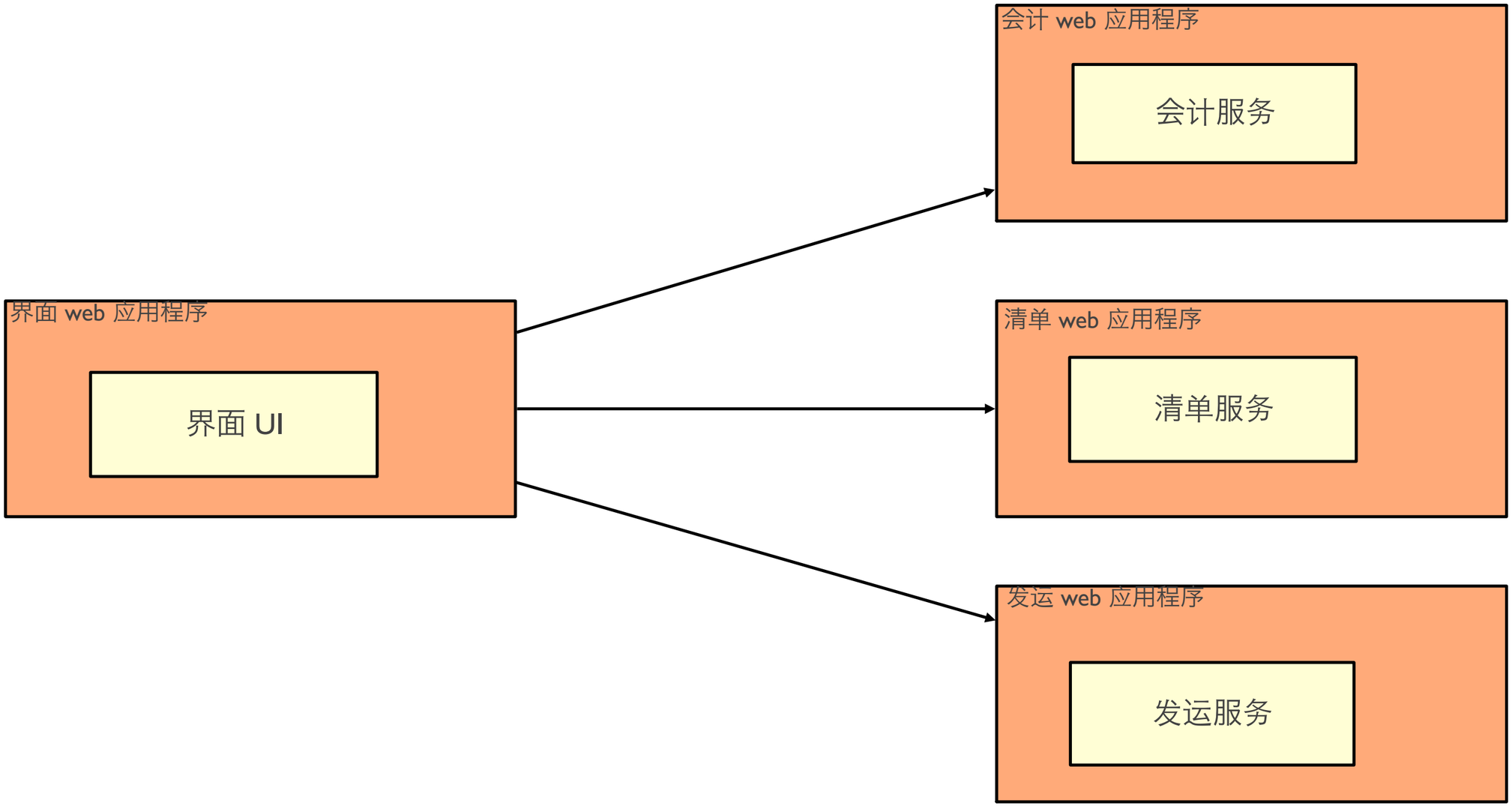
X 轴
- 水平复制

Z 轴 – 数据分区
通过对相似事务进行分割来扩展

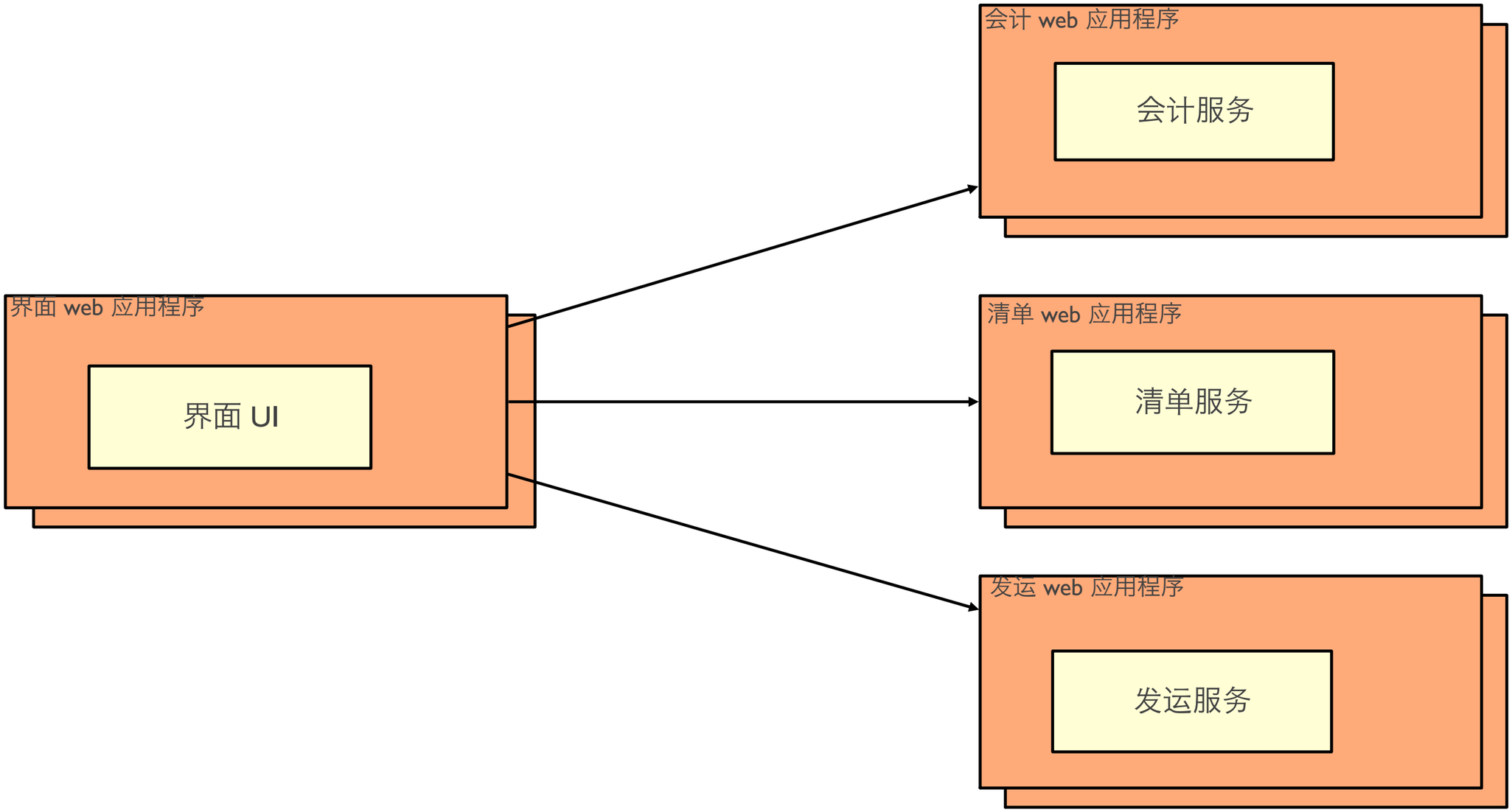
Y 轴扩展 – 应用程序级别



Y 轴扩展 – 应用程序级别


会计 web 应用程序
会计服务
界面 web 应用程序
界面 UI
清单 web 应用程序
清单服务
发运 web 应用程序
发运服务

Y 轴扩展 – 应用程序级别



对每种服务应用 X 轴复制和/或 Z 轴分区

分区策略

分区策略

- 采用动词分区，例如发运服务

分区策略

- 采用动词分区，例如发运服务
- 采用名词分区，例如清单服务

分区策略

- 采用动词分区，例如发运服务
- 采用名词分区，例如清单服务
- 单一负责制原则

分区策略

- 采用动词分区，例如发运服务
- 采用名词分区，例如清单服务
- 单一负责制原则
- Unix 实用程序 – 专注做好一件事

分区策略

- 采用动词分区，例如发运服务
- 采用名词分区，例如清单服务
- 单一负责制原则
- Unix 实用程序 – 专注做好一件事

一门艺术

实际案例

NETFLIX

http://techblog.netflix.com/

amazon

构建一个页面来访问 100-150 项服务。

http://highscalability.com/amazon-architecture

ebay Tech Blog

http://www.addsimplicity.com/downloads/eBaySDForum2006-11-29.pdf

http://queue.acm.org/detail.cfm?id=1394128

但弊端不可回避

复杂性

复杂性

请参阅 Steve Yegge 的文章《Google Platforms Rant》Amazon.com

多个数据库

=

事务管理挑战

初期：

- 不需要
- 会降低速度

何时使用？

初期：

- 不需要
- 会降低速度

后期：

- 需要
- 重构十分痛苦

何时使用?

初期：

- 不需要
- 会降低速度

后期：

- 需要
- 重构十分痛苦

但也有诸多优势

但也有诸多优势

- 扩展开发：可独立开发、部署和扩展各项服务

但也有诸多优势

- 扩展开发：可独立开发、部署和扩展各项服务
- 可单独升级 UI

但也有诸多优势

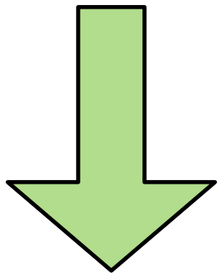
- 扩展开发：可独立开发、部署和扩展各项服务
- 可单独升级 UI
- 改进故障隔离

但也有诸多优势

- 扩展开发：可独立开发、部署和扩展各项服务
- 可单独升级 UI
- 改进故障隔离
- 消除对单一技术架构的长期依赖

但也有诸多优势

- 扩展开发：可独立开发、部署和扩展各项服务
- 可单独升级 UI
- 改进故障隔离
- 消除对单一技术架构的长期依赖



模块化、多语言、多框架应用程序

两级架构

系统级别

服务

服务之间的桥梁：接口和通信机制

变化缓慢

服务级别

各项服务的内部体系结构

各项服务可能采用不同的技术架构

为作业选择最适宜的工具

发展迅速

如果是小规模服务...

如果是小规模服务...

- 采用更优秀的技术架构定期重写

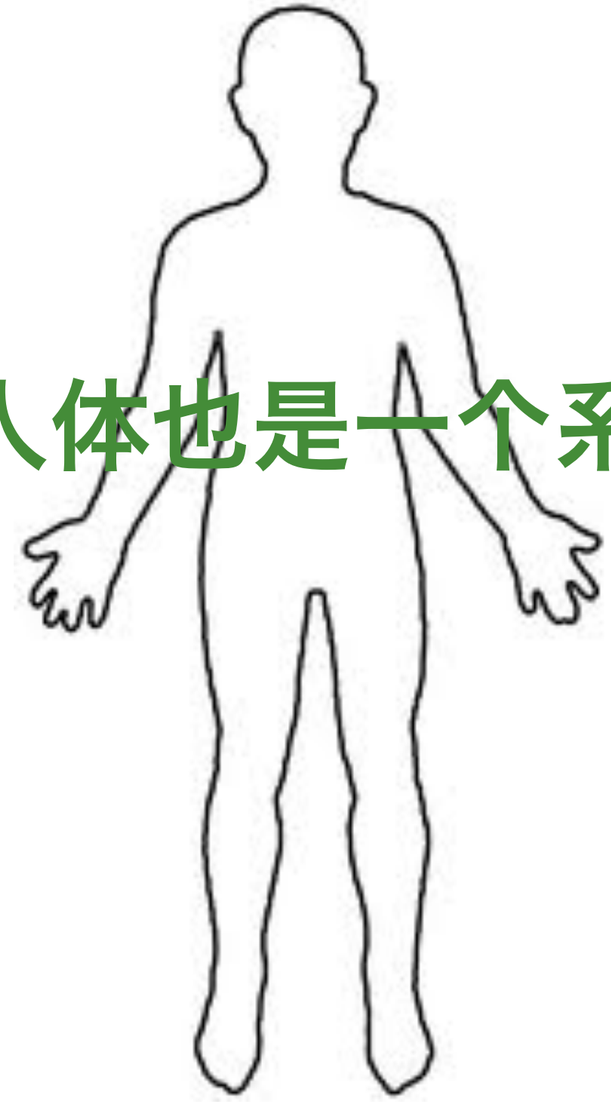
如果是小规模服务...

- 采用更优秀的技术架构定期重写
- 根据不断变化的要求和更优秀的技术来调整系统而无需全部重写

如果是小规模服务...

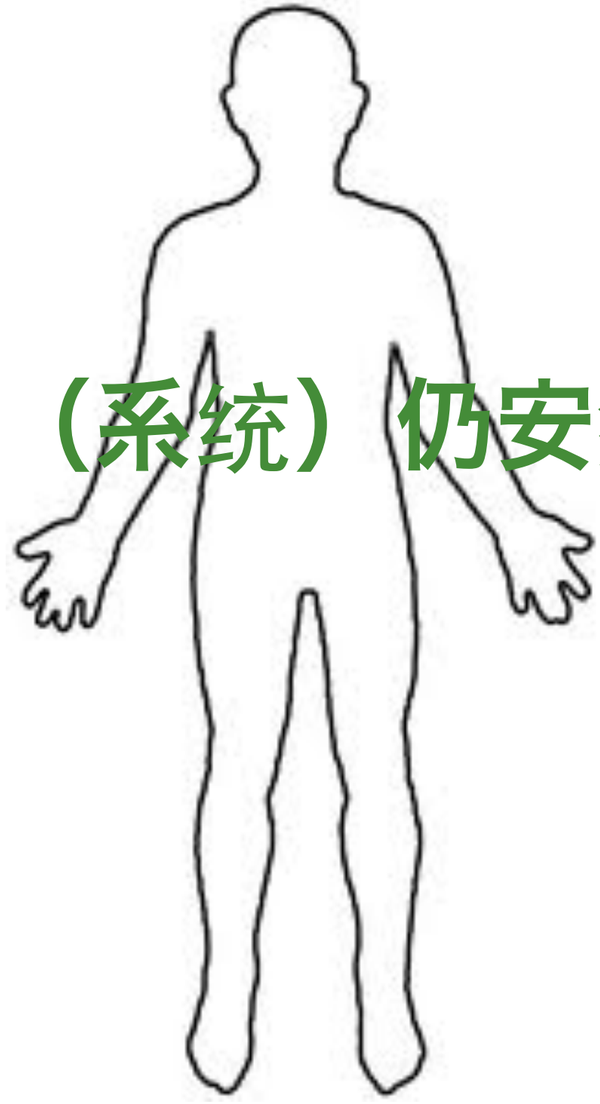
- 采用更优秀的技术架构定期重写
- 根据不断变化的要求和更优秀的技术来调整系统而无需全部重写
- 挑选最优秀的开发人员，而不是最优秀的**<选用一种语言>开发人员** ⇒ 多语言文化

人体也是一个系统



每天有 500 亿到 700 亿个细胞死亡

但您（系统）仍安然无恙



我们能否创建具有上述特点的软件系统？

我们能否创建具有上述特点的软件系统？

http://dreamsongs.com/Files/DesignBeyondHumanAbilitiesSimp.pdf

http://dreamsongs.com/Files/WhitherSoftware.pdf

议题

- 整体性（有时不如人意）
- 将应用程序分解为服务
- 服务如何通信？
- 表示层设计
- Cloud Foundry 有何帮助

服务间通信选项

- 同步 HTTP ⇔ 异步 AMQP
- 格式：JSON、XML、Protocol Buffers、Thrift ...
- 甚至通过数据库

服务间通信选项

- 同步 HTTP ⇔ 异步 AMQP
- 格式：JSON、XML、Protocol Buffers、Thrift ...
- 甚至通过数据库

首选异步

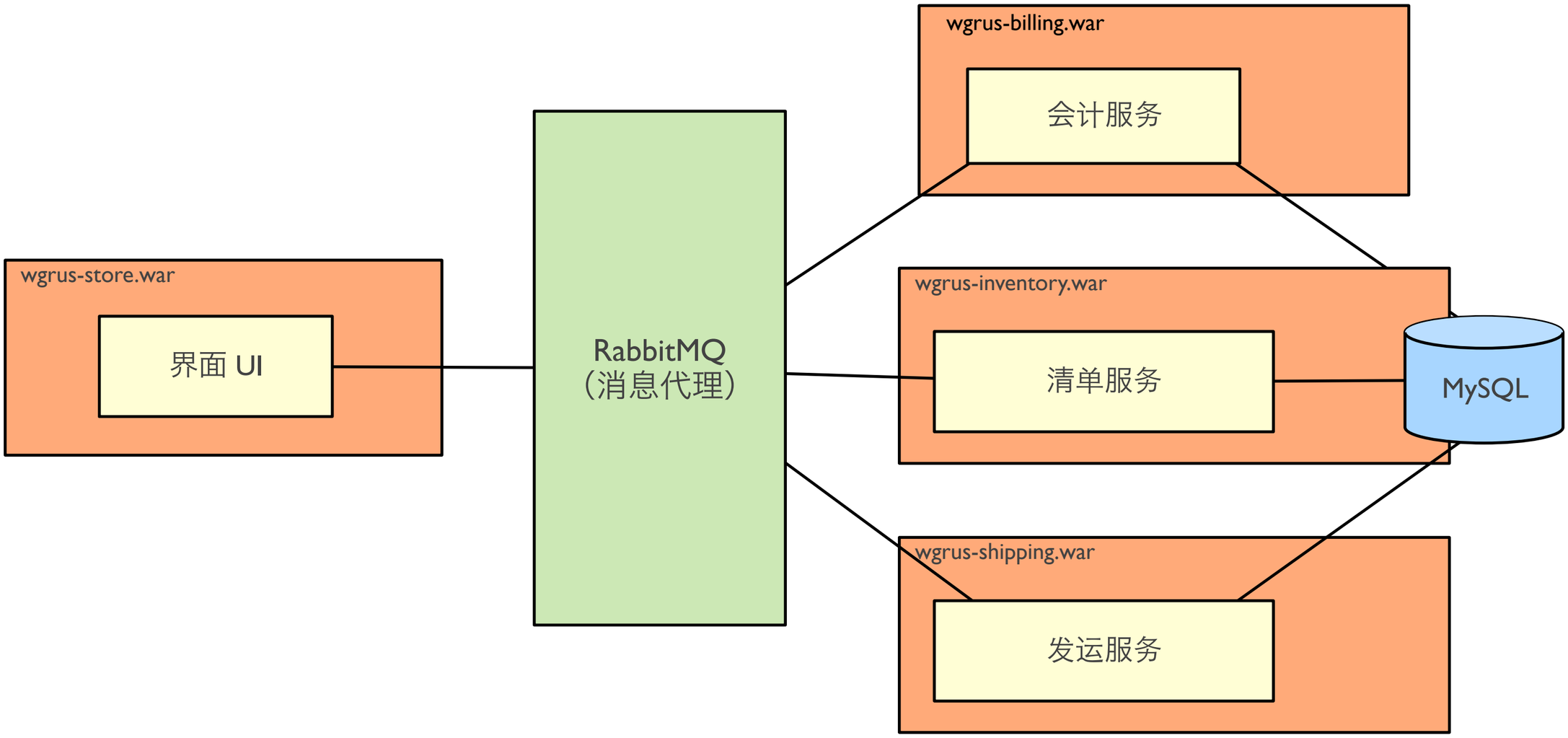
服务间通信选项

- 同步 HTTP ⇔ 异步 AMQP
- 格式：JSON、XML、Protocol Buffers、Thrift ...
- 甚至通过数据库

首选异步

虽然 JSON 很流行，但二进制格式更高效

基于异步消息的通信


wgrus-billing.war
会计服务
wgrus-store.war
界面 UI
RabbitMQ
(消息代理)
wgrus-inventory.war
清单服务
MySQL
wgrus-shipping.war
发运服务

优势

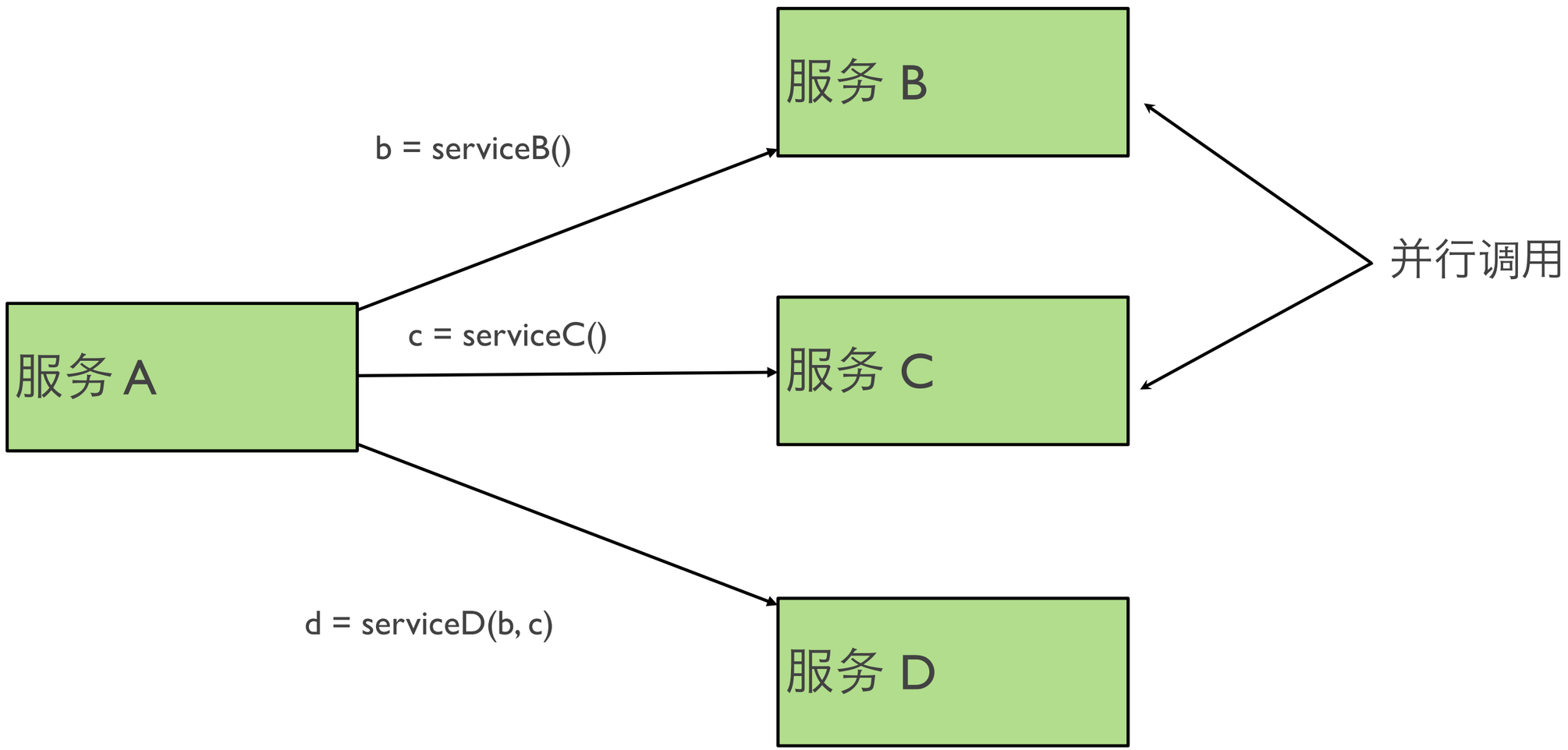
- 将调用程序从服务器解耦
- 调用程序不知道服务器的坐标 (URL)
- 服务器宕机/缓慢时，消息代理会缓冲消息
- 支持各种通信模式，如点对点、发布-订阅 ...

弊端

- 消息代理带来的额外复杂性
- 请求/回复式通信更为复杂

编写调用服务的代码

需要并行



Java Futures为高度并发抽象

http://en.wikipedia.org/wiki/Futures_and_promises

使用 Java Futures

```
public class Client {

    private ExecutorService executorService;
    private RemoteServiceProxy remoteServiceProxy;

    public void doSomething() throws ... {
        Future<Integer> result =
          executorService.submit(new Callable<Integer>() {
            @Override
            public Integer call() throws Exception {
                return remoteServiceProxy.invokeRemoteService();
            }
        });

        /// Do other things

        int r = result.get(500, TimeUnit.MILLISECONDS);

        System.out.println(r);
```

使用 Java Futures

```
public class Client {

    private ExecutorService executorService;
    private RemoteServiceProxy remoteServiceProxy;

    public void doSomething() throws ... {
        Future<Integer> result =
          executorService.submit(new Callable<Integer>() {
            @Override
            public Integer call() throws Exception {
                return remoteServiceProxy.invokeRemoteService();
            }
        });

        /// Do other things

        int r = result.get(500, TimeUnit.MILLISECONDS);

        System.out.println(r);
```

最终包含结果

使用 Java Futures

```
public class Client {

    private ExecutorService executorService;
    private RemoteServiceProxy remoteServiceProxy;

    public void doSomething() throws ... {
        Future<Integer> result =
          executorService.submit(new Callable<Integer>() {
            @Override
            public Integer call() throws Exception {
                return remoteServiceProxy.invokeRemoteService();
            }
        });

        /// Do other things

        int r = result.get(500, TimeUnit.MILLISECONDS);

        System.out.println(r);
```

最终包含结果

需要时等待结果

Akka 的可组合Futures更佳

可组合Future

```
val f1 = Future {  ... ; 1 }
val f2 = Future {  ... ; 2 }



val f4 = f2.map(_ * 2)
assertEquals(4, Await.result(f4, 1 second))

val fzip = f1 zip f2
assertEquals((1, 2), Await.result(fzip,
```

http://doc.akka.io/docs/akka/2.0.1/scala/futures.html

可组合Future

```
val f1 = Future {  ... ; 1 }
val f2 = Future {  ... ; 2 }


val f4 = f2.map(_ * 2)
assertEquals(4, Await.result(f4, 1 second))

val fzip = f1 zip f2
assertEquals((1, 2), Await.result(fzip,
```

转换Future

http://doc.akka.io/docs/akka/2.0.1/scala/futures.html

可组合Future

```
val f1 = Future {  ... ; 1 }
val f2 = Future {  ... ; 2 }


val f4 = f2.map(_ * 2)
assertEquals(4, Await.result(f4, 1 second))

val fzip = f1 zip f2
assertEquals((1, 2), Await.result(fzip,
```

转换Future

组合两项Future

http://doc.akka.io/docs/akka/2.0.1/scala/futures.html

采用 Akka futures

```
def callB() : Future[...] = ...
def callC() : Future[...] = ...
def callD() : Future[...] = ...

val future = for {
  (b, c) <- callB() zip callC();
  d <- callD(b, c)
 } yield d

val result = Await.result(future, 1 second)
```

http://doc.akka.io/docs/akka/2.0.1/scala/futures.html

采用 Akka futures

```
def callB() : Future[...] = ...
def callC() : Future[...] = ...
def callD() : Future[...] = ...

val future = for {
  (b, c) <- callB() zip callC();
  d <- callD(b, c)
 } yield d

val result = Await.result(future, 1 second)
```

两个调用并行执行

http://doc.akka.io/docs/akka/2.0.1/scala/futures.html

采用 Akka futures

```
def callB() : Future[...] = ...
def callC() : Future[...] = ...
def callD() : Future[...] = ...

val future = for {
  (b, c) <- callB() zip callC();
  d <- callD(b, c)
 } yield d

val result = Await.result(future, 1 second)
```

两个调用并行执行

然后调用 D

http://doc.akka.io/docs/akka/2.0.1/scala/futures.html

采用 Akka futures

```
def callB() : Future[...] = ...
def callC() : Future[...] = ...
def callD() : Future[...] = ...

val future = for {
  (b, c) <- callB() zip callC();
  d <- callD(b, c)
 } yield d

val result = Await.result(future, 1 second)
```

两个调用并行执行

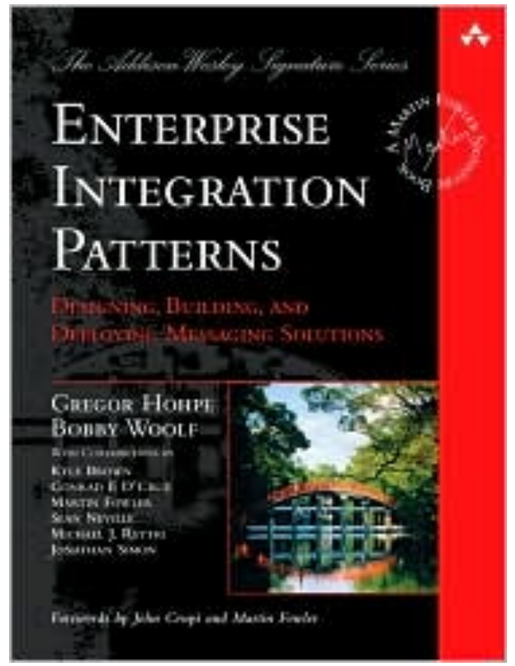
然后调用 D

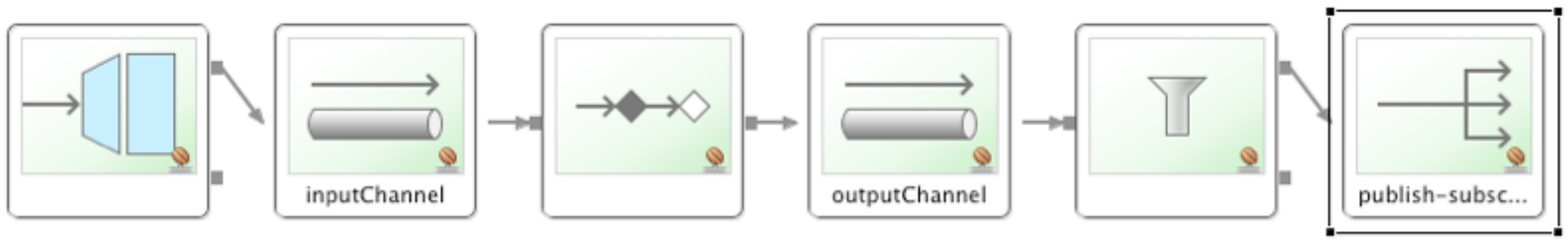
获得 D 的结果

http://doc.akka.io/docs/akka/2.0.1/scala/futures.html

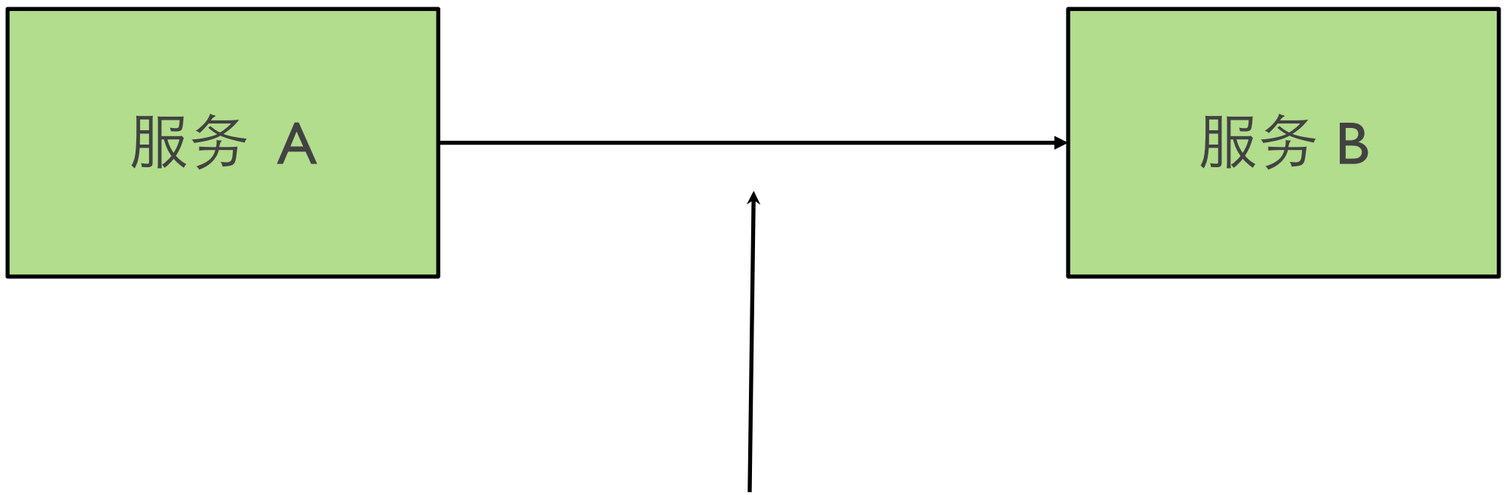
Spring 集成

- 为管道和筛选器体系结构提供构建块
- 支持开发具有下述特征的应用组件
  - 松耦合
  - 与消息传送基础架构隔绝
- 明确定义消息传送





处理故障



分布式系统中发生错误

关于 Netflix

关于 Netflix

> 1B API 调用/天

关于 Netflix

> 1B API 调用/天

1 API 调用 ⇒ 平均 6 次服务调用

关于 Netflix

> 1B API 调用/天

1 API 调用 ⇒ 平均 6 次服务调用

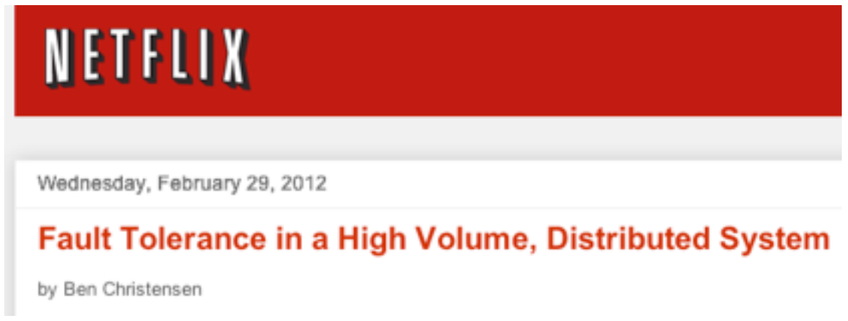
容错性必不可少

关于 Netflix

> 1B API 调用/天

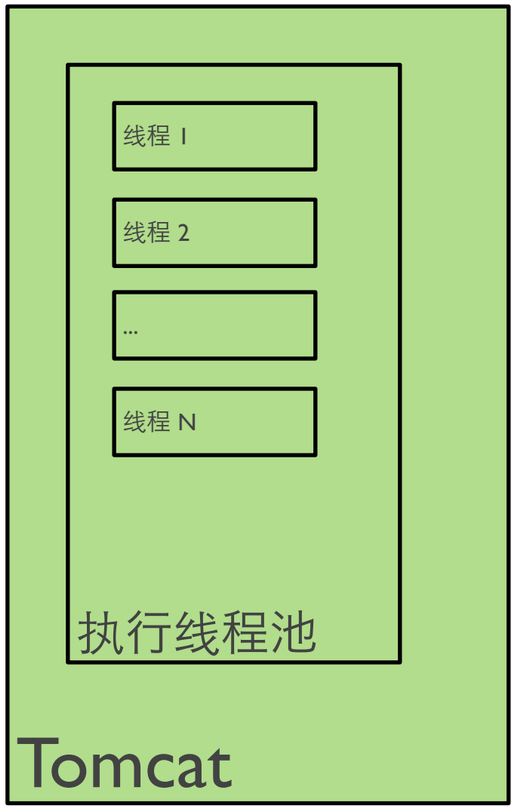
1 API 调用 ⇒ 平均 6 次服务调用

容错性必不可少

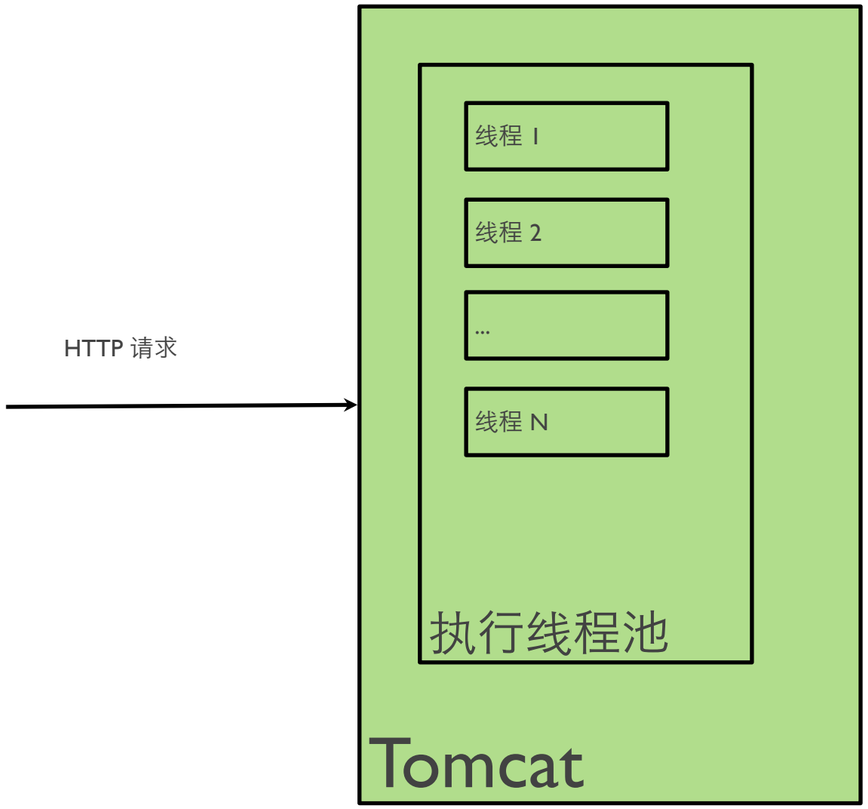


http://techblog.netflix.com/2012/02/fault-tolerance-in-high-volume.html

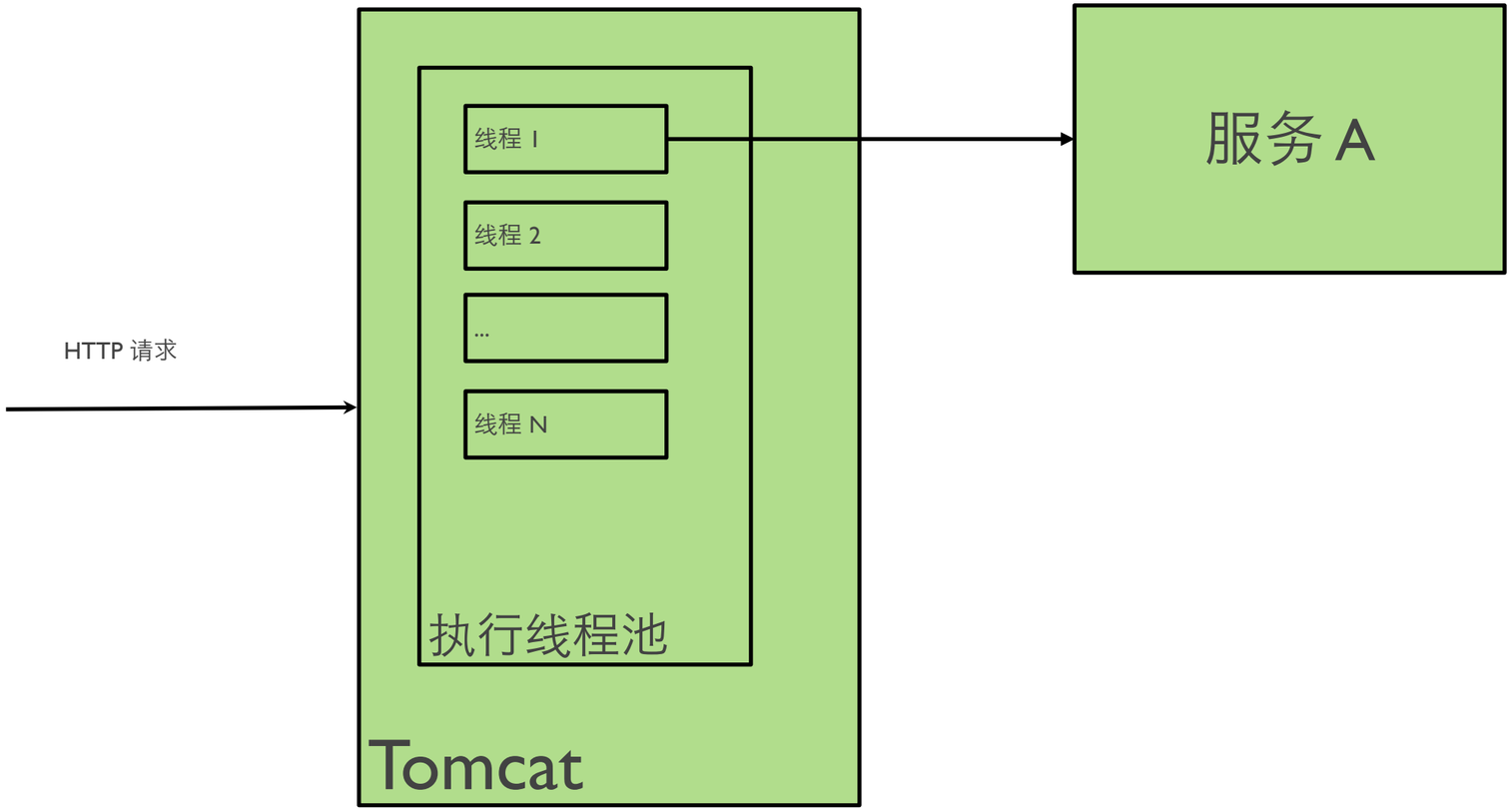
如何耗尽线程



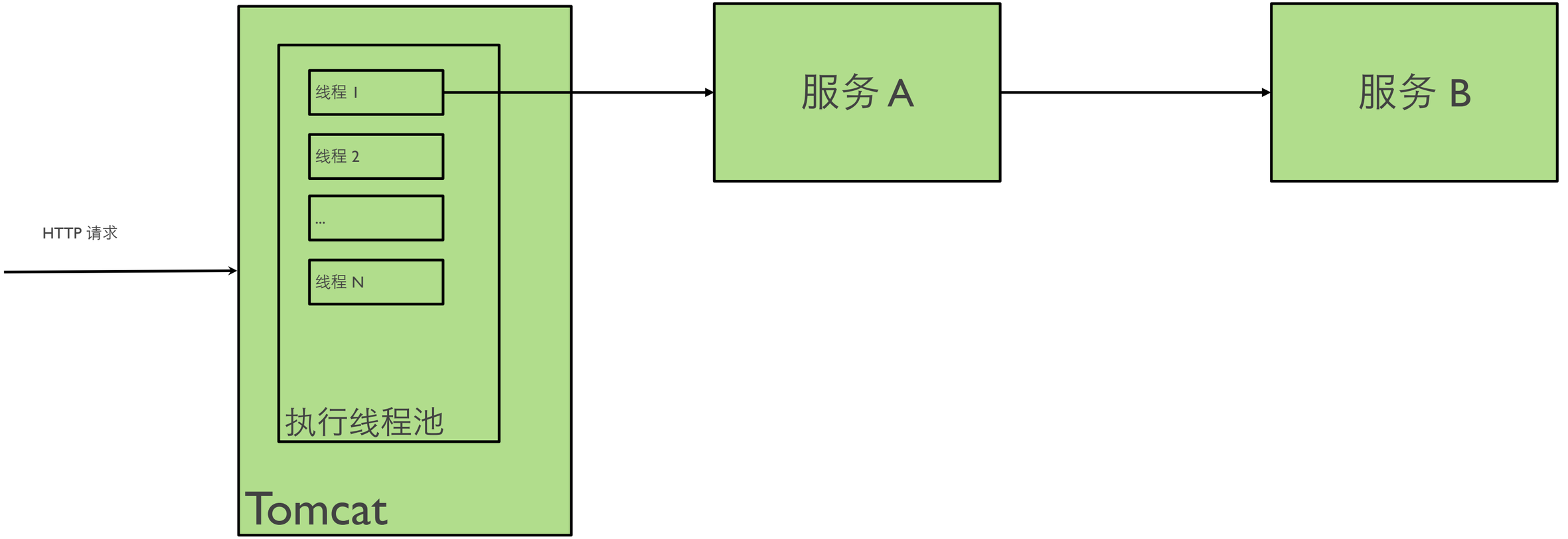
如何耗尽线程



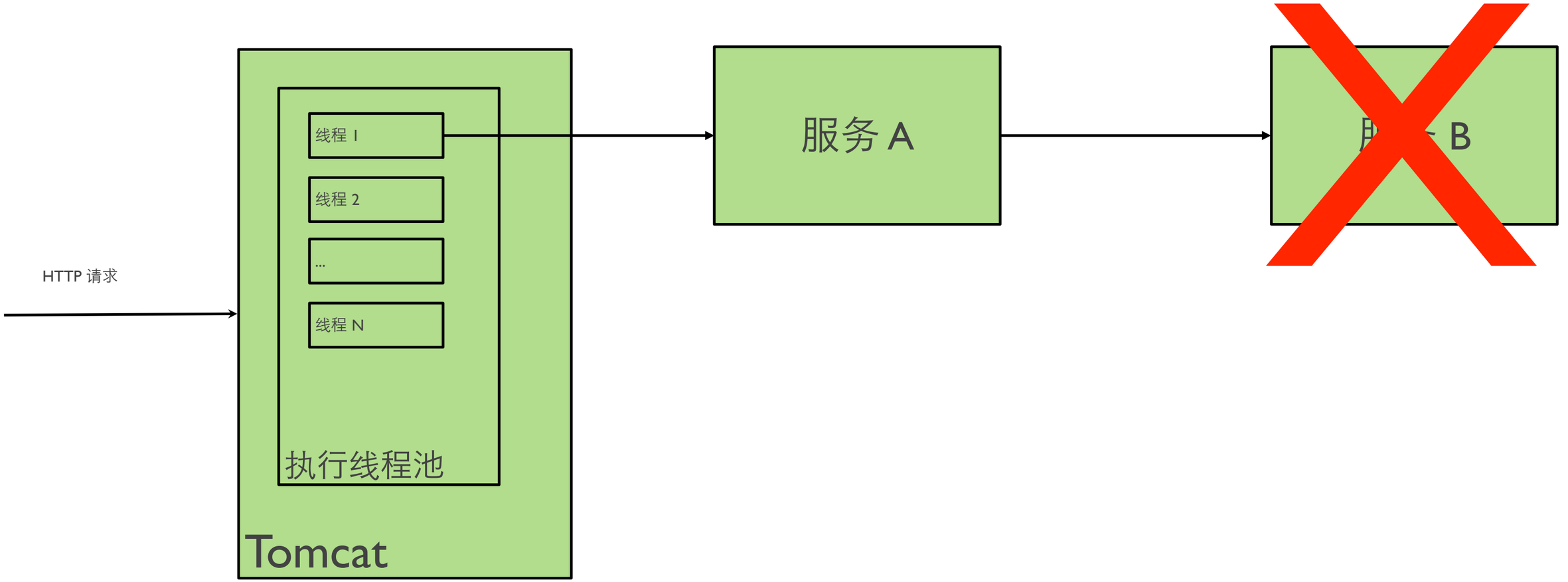
如何耗尽线程



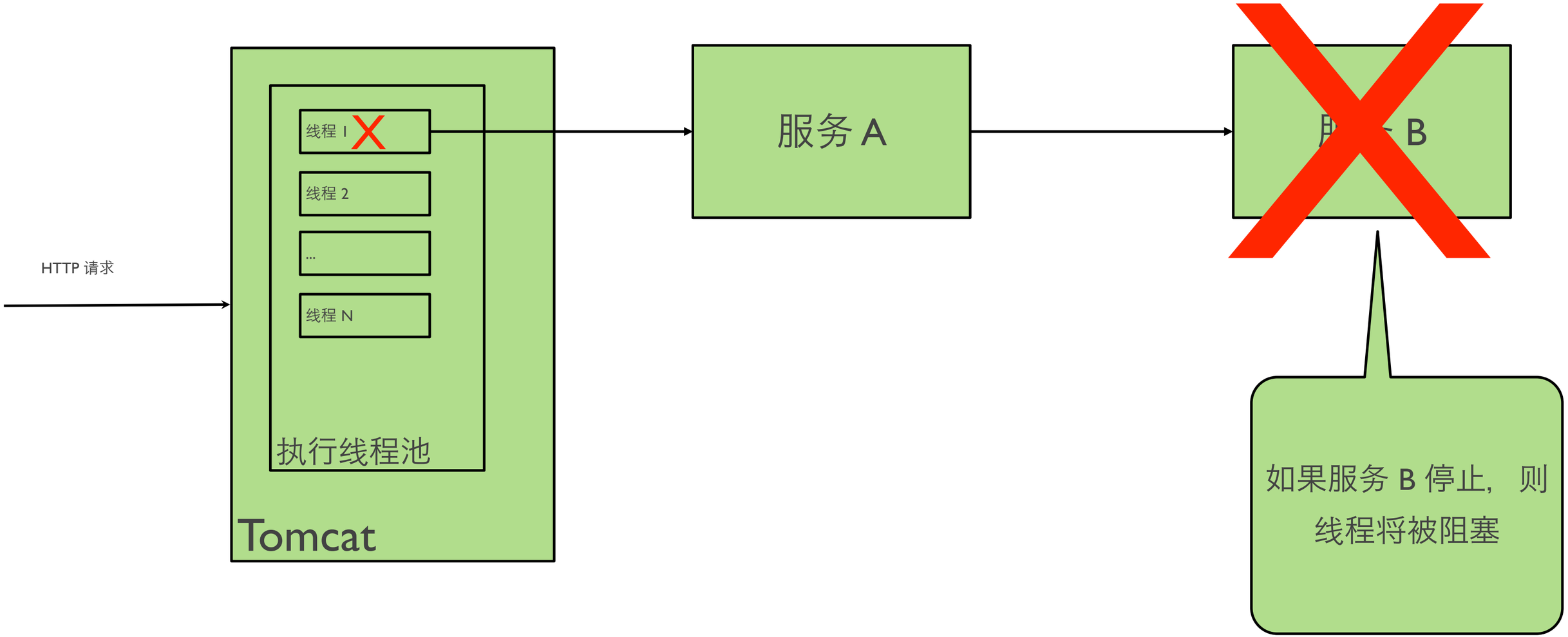
如何耗尽线程



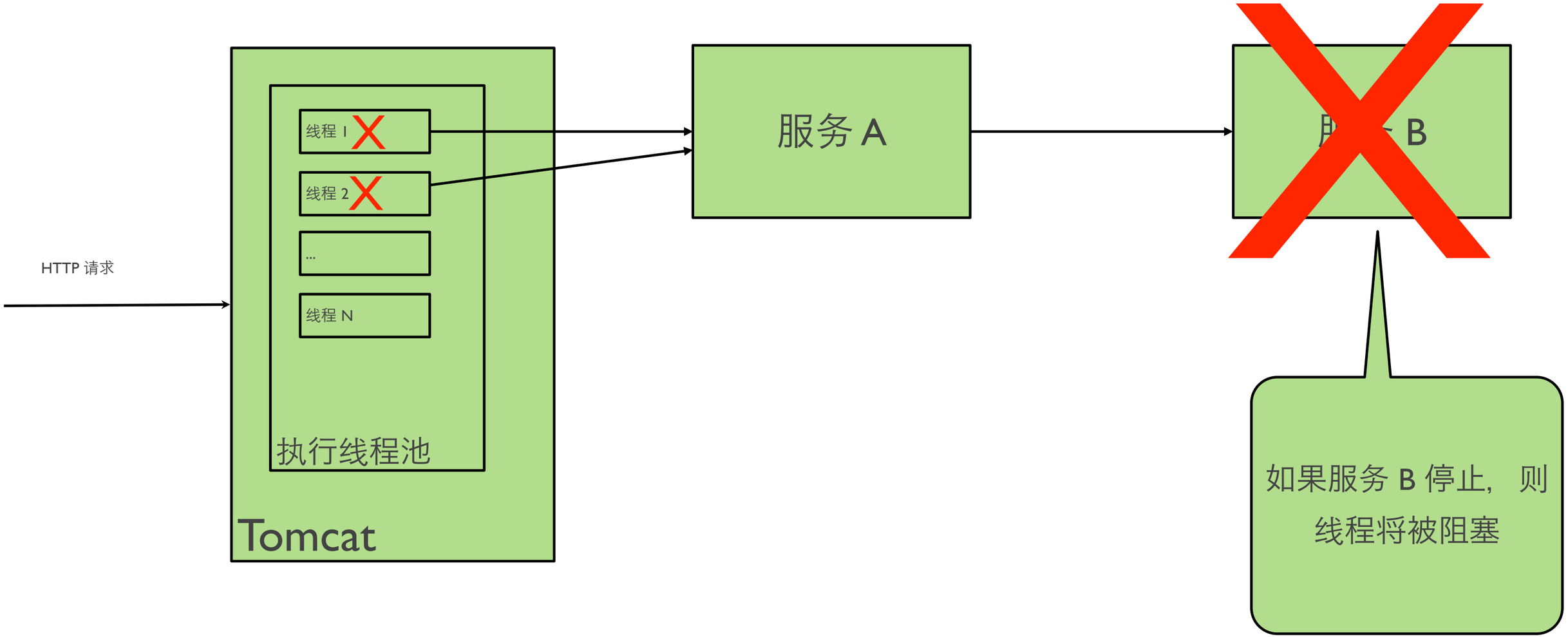
如何耗尽线程



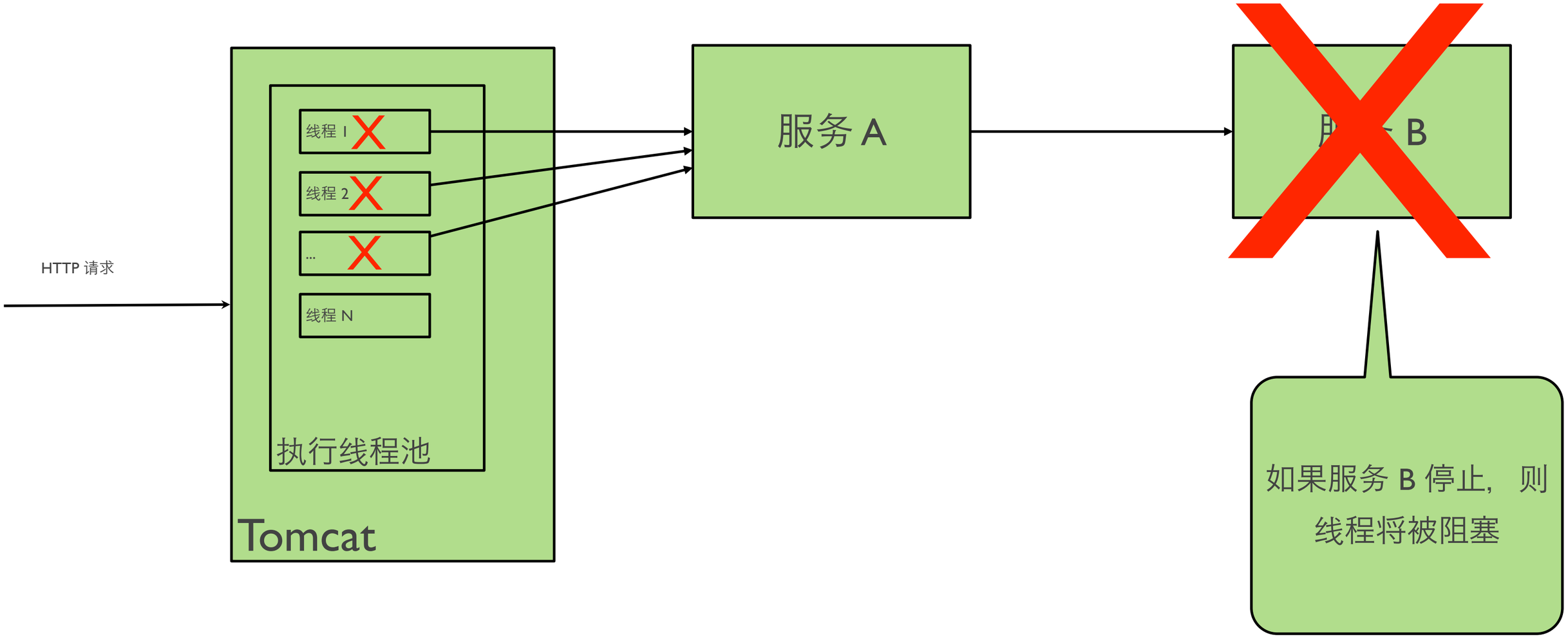
如何耗尽线程



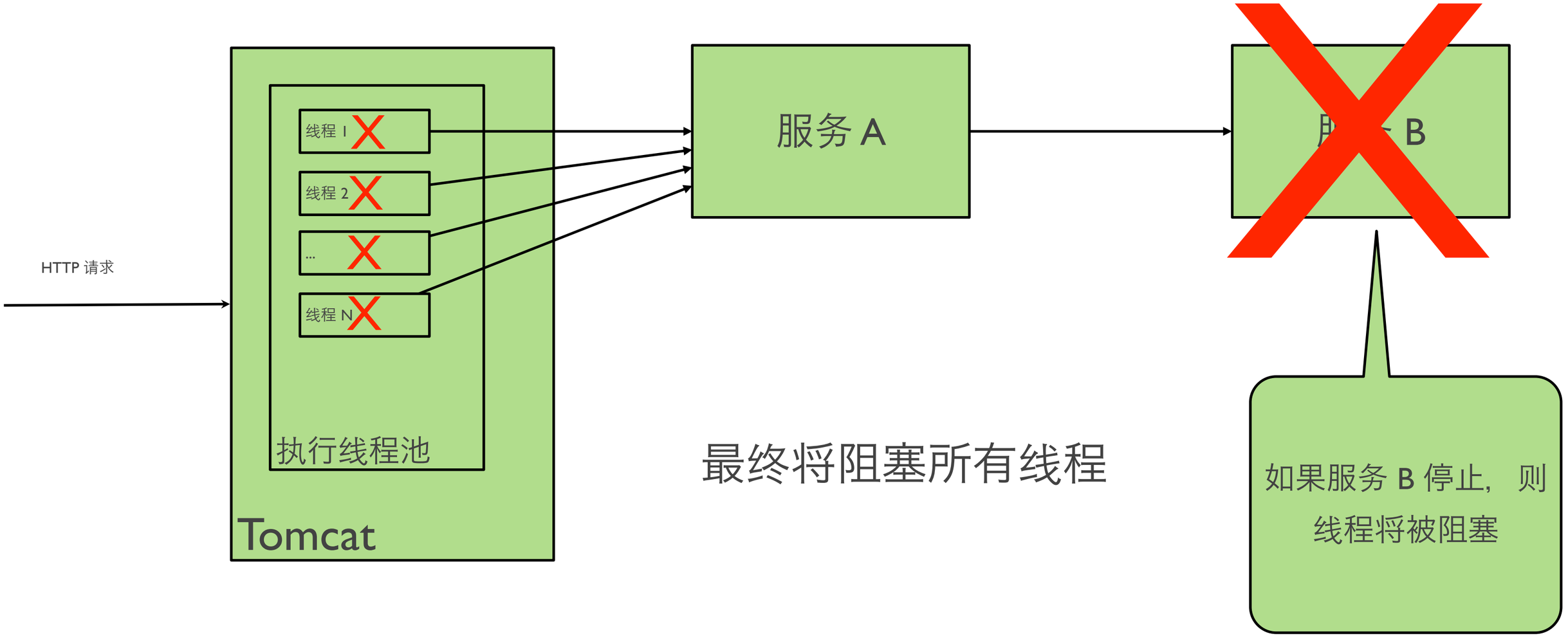
如何耗尽线程



如何耗尽线程



如何耗尽线程



最终将阻塞所有线程

使用超时与重试

NETFLIX

Wednesday, February 29, 2012

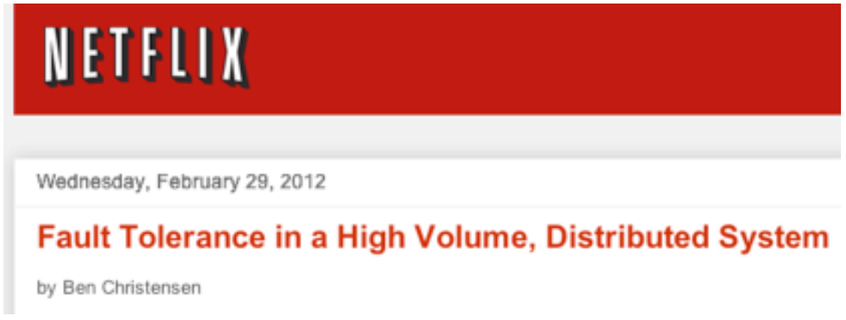
Fault Tolerance in a High Volume, Distributed System

by Ben Christensen

http://techblog.netflix.com/2012/02/fault-tolerance-in-high-volume.html

使用超时与重试

不必耗时等待



http://techblog.netflix.com/2012/02/fault-tolerance-in-high-volume.html

使用超时与重试

不必耗时等待

暂时性错误 ⇒ 重试

NETFLIX

Wednesday, February 29, 2012

Fault Tolerance in a High Volume, Distributed System

by Ben Christensen

http://techblog.netflix.com/2012/02/fault-tolerance-in-high-volume.html

使用按依赖项绑定的线程池

服务 A

服务 B

NETFLIX

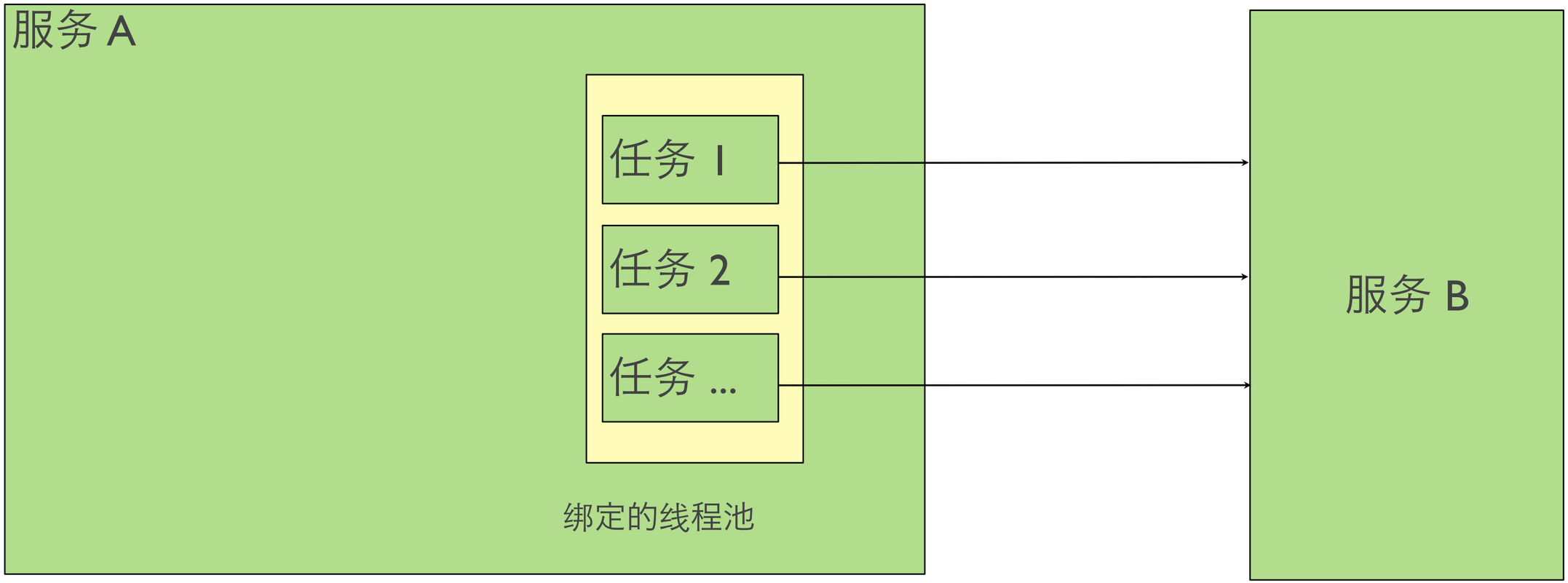
Wednesday, February 29, 2012

Fault Tolerance in a High Volume, Distributed System

by Ben Christensen

http://techblog.netflix.com/2012/02/fault-tolerance-in-high-volume.html

使用按依赖项绑定的线程池




NETFLIX

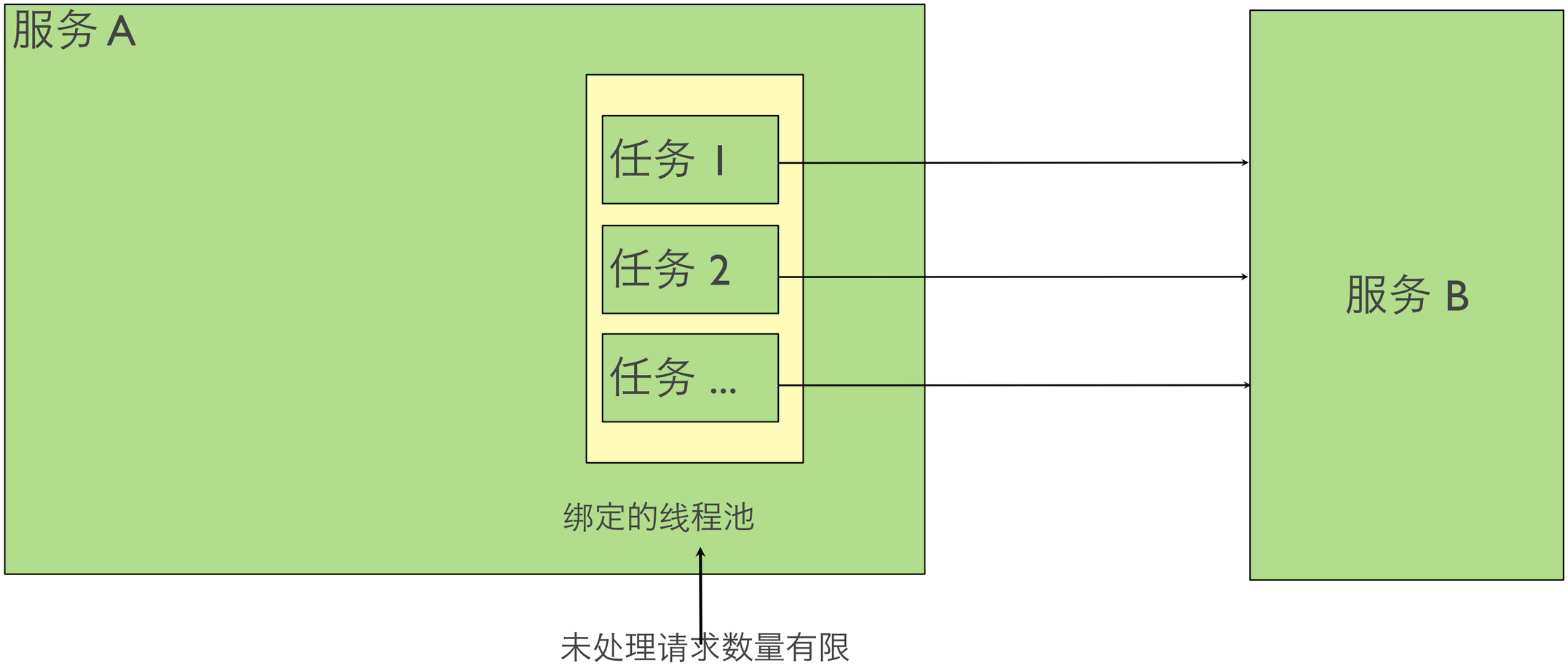
Wednesday, February 29, 2012

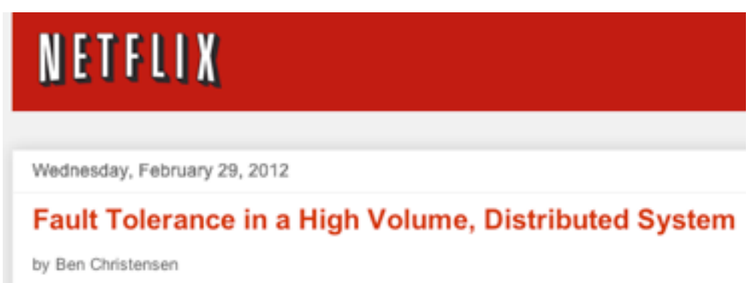
Fault Tolerance in a High Volume, Distributed System

by Ben Christensen

http://techblog.netflix.com/2012/02/fault-tolerance-in-high-volume.html

使用按依赖项绑定的线程池




NETFLIX

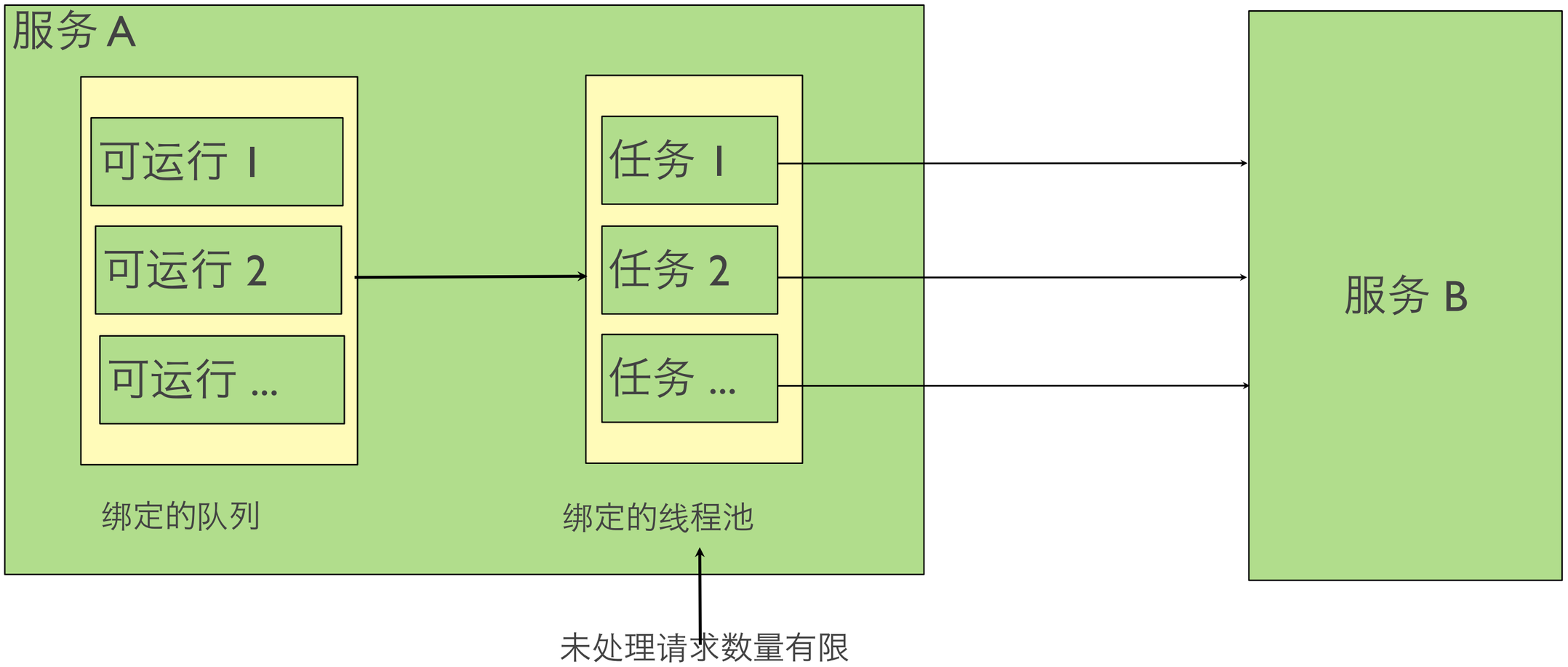
Wednesday, February 29, 2012

Fault Tolerance in a High Volume, Distributed System

by Ben Christensen

http://techblog.netflix.com/2012/02/fault-tolerance-in-high-volume.html

使用按依赖项绑定的线程池



NETFLIX

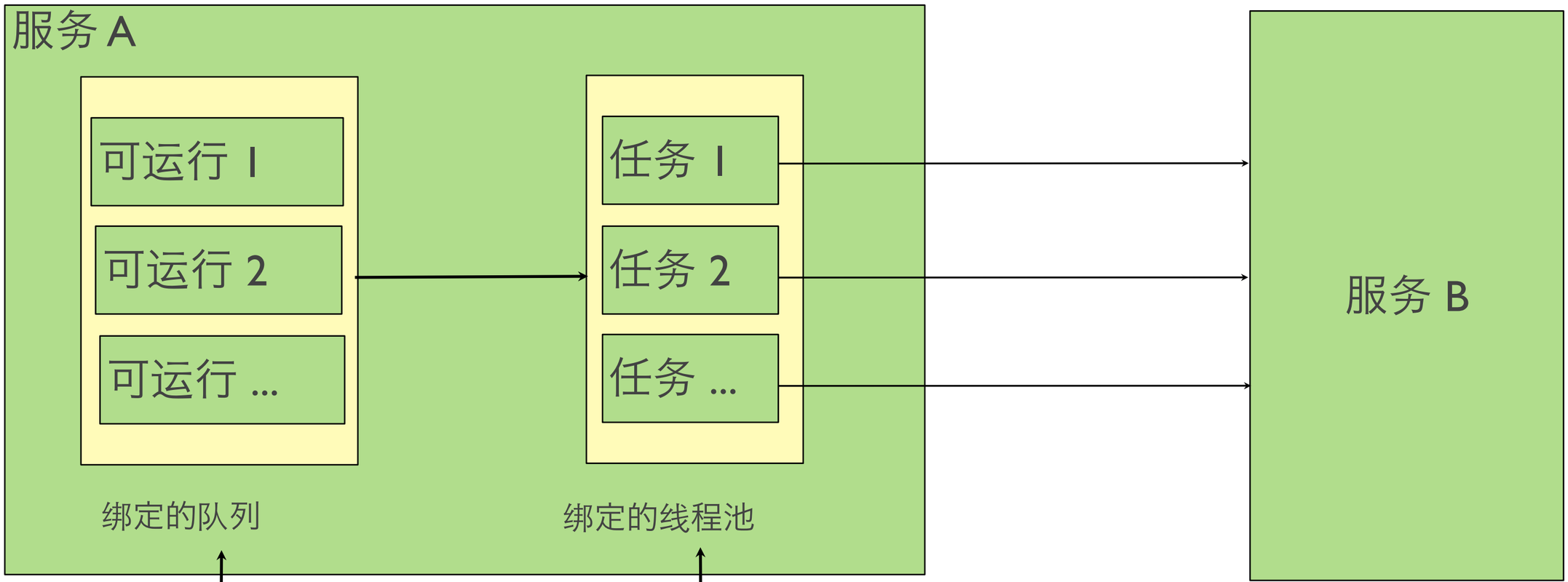
Wednesday, February 29, 2012

Fault Tolerance in a High Volume, Distributed System

by Ben Christensen

http://techblog.netflix.com/2012/02/fault-tolerance-in-high-volume.html

使用按依赖项绑定的线程池



如果服务缓慢或停止，
则会很快出现故障

未处理请求数量有限

NETFLIX

Wednesday, February 29, 2012

Fault Tolerance in a High Volume, Distributed System

by Ben Christensen

http://techblog.netflix.com/2012/02/fault-tolerance-in-high-volume.html

使用断路器

NETFLIX

Wednesday, February 29, 2012

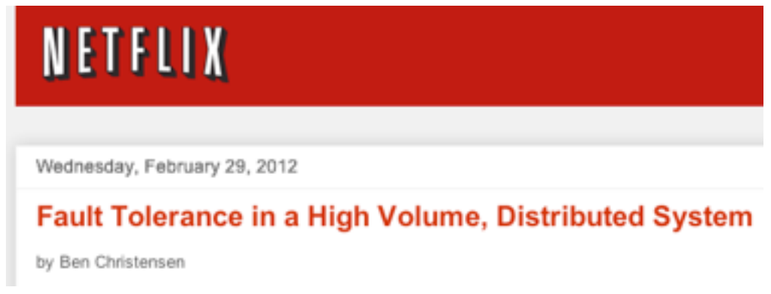
Fault Tolerance in a High Volume, Distributed System

by Ben Christensen

http://techblog.netflix.com/2012/02/fault-tolerance-in-high-volume.html

使用断路器

高错误率 ⇒ 暂时停止调用

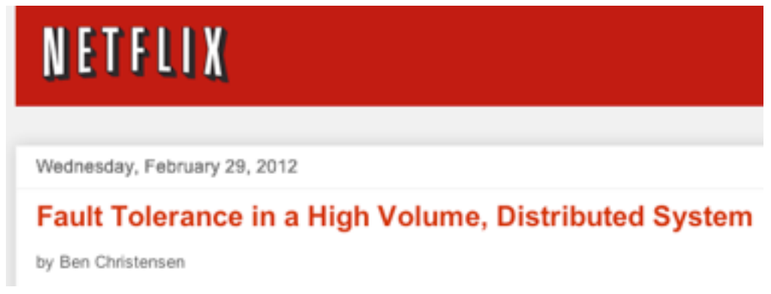


http://techblog.netflix.com/2012/02/fault-tolerance-in-high-volume.html

使用断路器

高错误率 ⇒ 暂时停止调用

停止 ⇒ 等待恢复



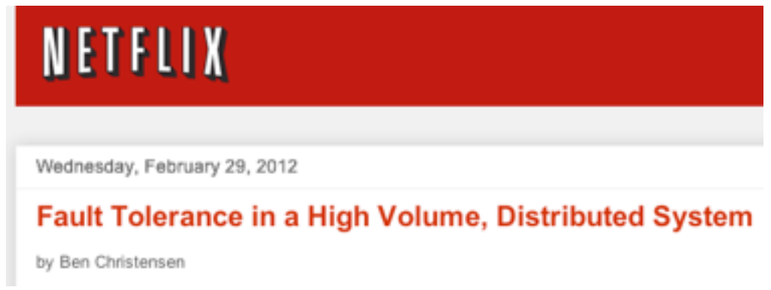
http://techblog.netflix.com/2012/02/fault-tolerance-in-high-volume.html

使用断路器

高错误率 ⇒ 暂时停止调用

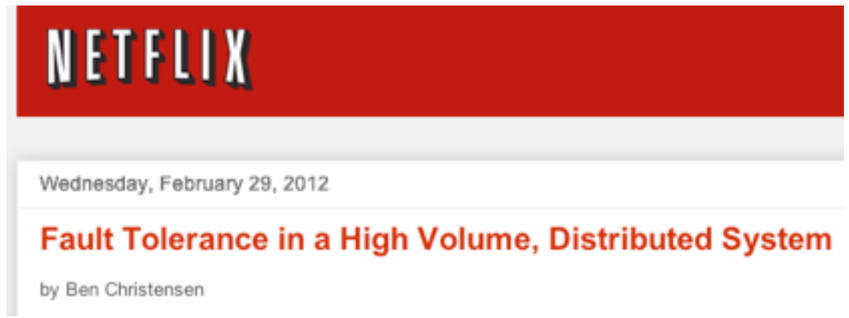
停止 ⇒ 等待恢复

缓慢 ⇒ 提供恢复正常的机会



http://techblog.netflix.com/2012/02/fault-tolerance-in-high-volume.html

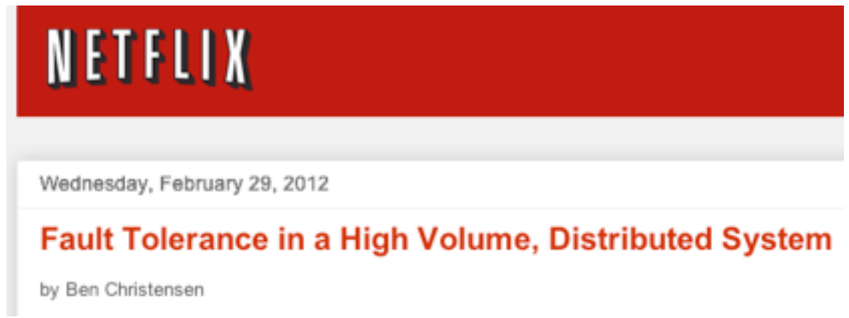
关于故障



http://techblog.netflix.com/2012/02/fault-tolerance-in-high-volume.html

关于故障

返回缓存数据

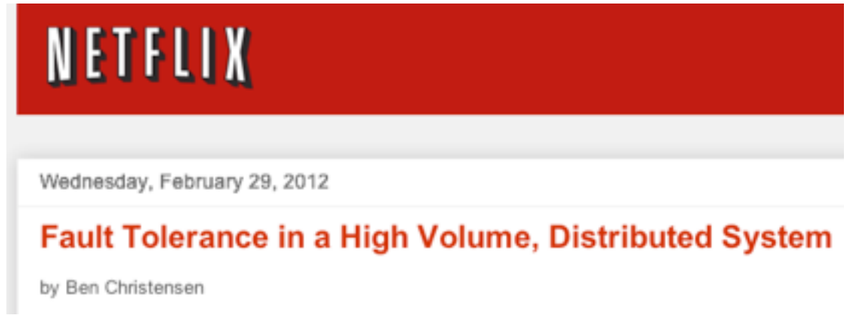


http://techblog.netflix.com/2012/02/fault-tolerance-in-high-volume.html

关于故障

返回缓存数据

返回默认数据



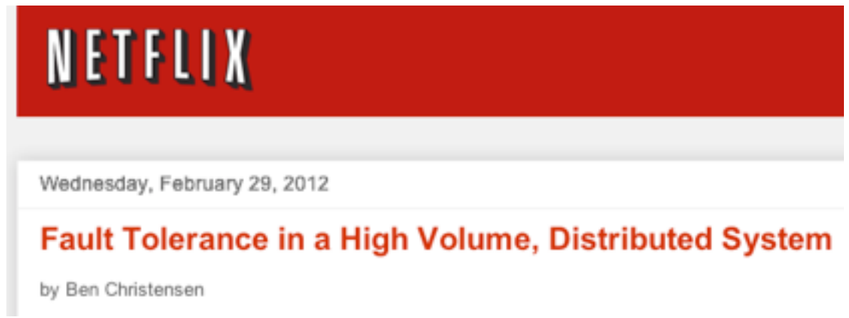
http://techblog.netflix.com/2012/02/fault-tolerance-in-high-volume.html

关于故障

返回缓存数据

返回默认数据

快速出现故障



http://techblog.netflix.com/2012/02/fault-tolerance-in-high-volume.html

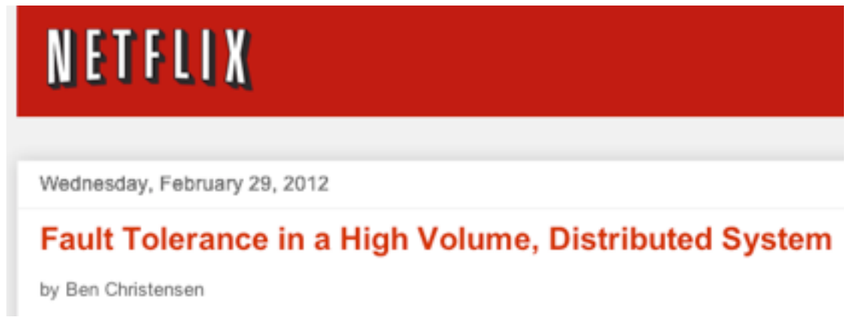
关于故障

避免故障

返回缓存数据

返回默认数据

快速出现故障

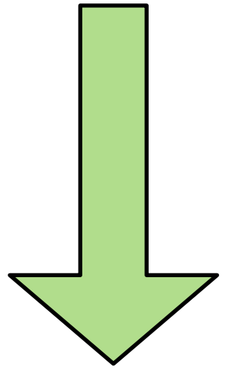


http://techblog.netflix.com/2012/02/fault-tolerance-in-high-volume.html

议题

- 整体性（有时不尽人意）
- 将应用程序分解为服务
- 服务如何通信？
- 表示层设计
- Cloud Foundry 有何帮助

模块化应用程序



NodeJS 是流行技术

为什么选择 NodeJS？

为什么选择 NodeJS？

- 与 Javascript 相似

为什么选择 NodeJS？

- 与 Javascript 相似
- 高性能、可扩展的事件驱动、非阻塞 I/O 模型

为什么选择 NodeJS？

- 与 Javascript 相似
- 高性能、可扩展的事件驱动、非阻塞 I/O 模型
- 紧凑的运行时

为什么选择 NodeJS？

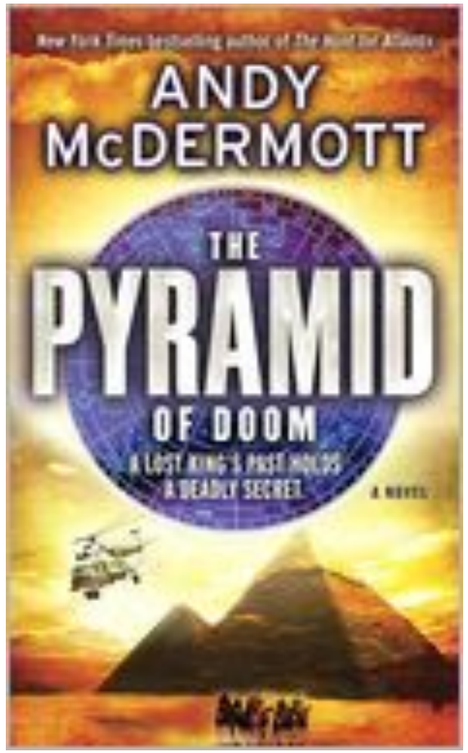
- 与 Javascript 相似
- 高性能、可扩展的事件驱动、非阻塞 I/O 模型
- 紧凑的运行时
- 社区开发的模块超过 17,000 个

为什么选择 NodeJS？

- 与 Javascript 相似
- 高性能、可扩展的事件驱动、非阻塞 I/O 模型
- 紧凑的运行时
- 社区开发的模块超过 17,000 个
- 许多 JavaScript 客户端框架均有类似 NodeJS 的组件，如 socket.io 和 SockJS

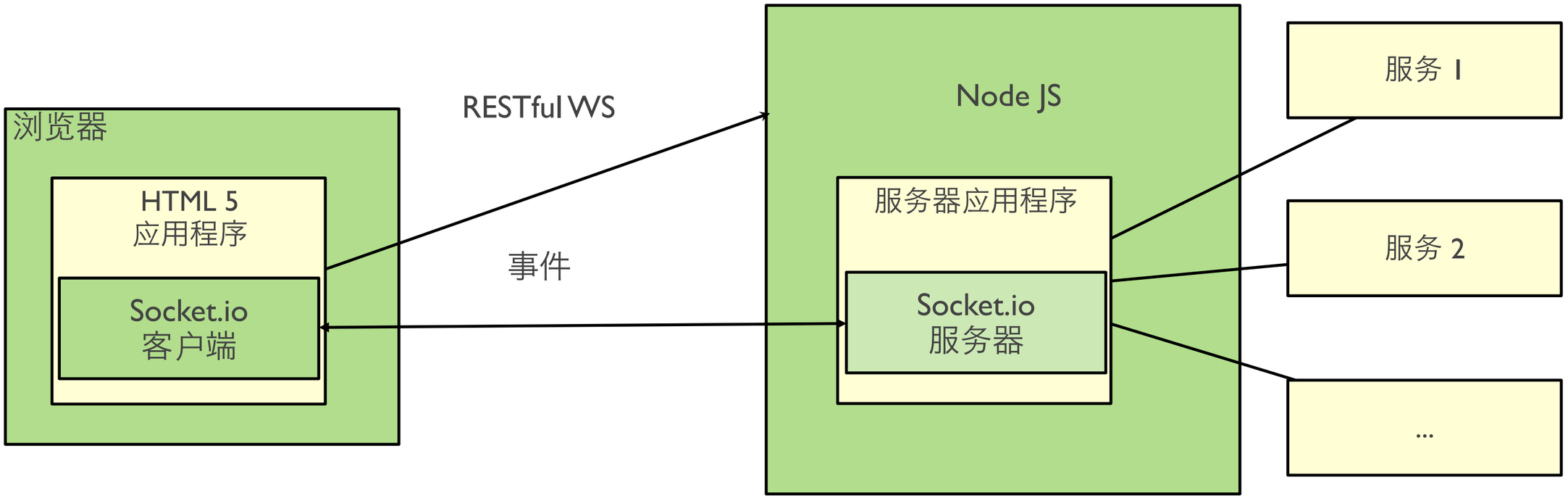
为什么不选择 NodeJS？

为什么不选择 NodeJS？

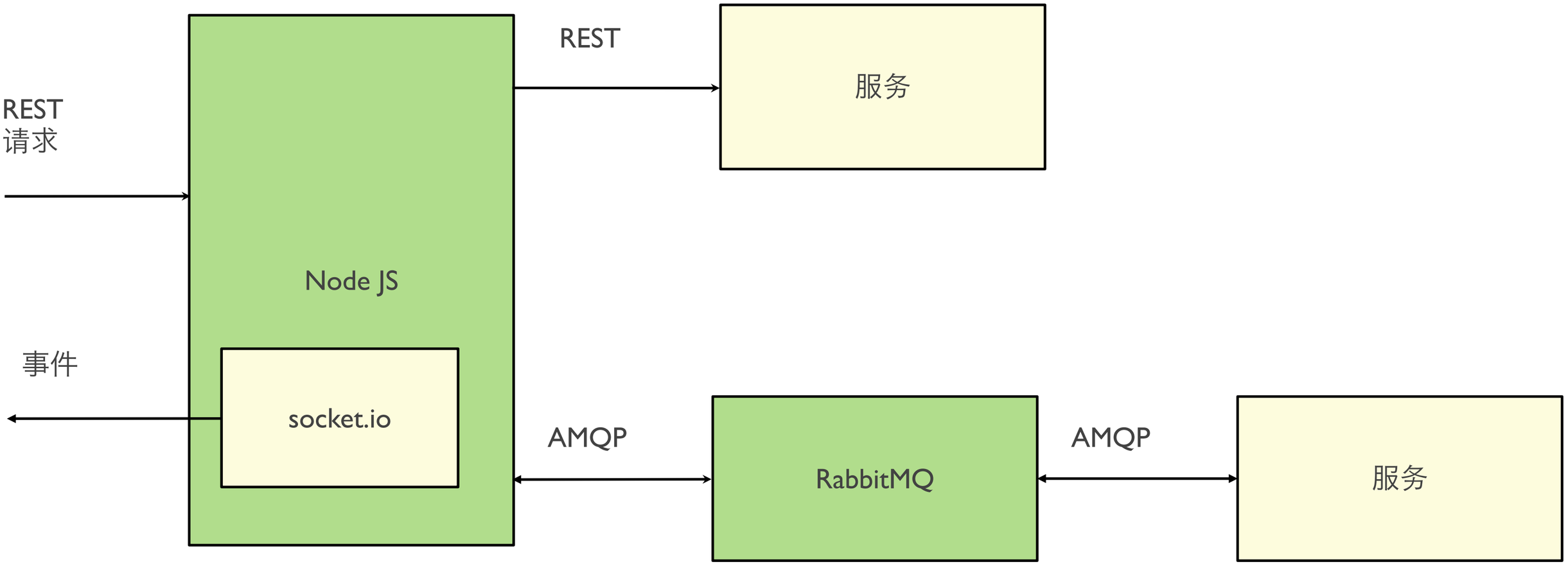


a.k.a. callback hell

一款现代 Web 应用程序



NodeJS – 使用 RESTful WS 和 AMQP



Socket.io – 服务器端

```
var express = require('express')
   , http = require('http')
   , amqp = require('amqp')
   ....;

server.listen(8081);
...
var amqpCon = amqp.createConnection(...);

io.sockets.on('connection', function (socket) {

  function amqpMessageHandler(message, headers, deliveryInfo) {
     var m = JSON.parse(message.data.toString());
     socket.emit('tick', m);
   };
   amqpCon.queue("", {},
      function(queue) {
         queue.bind("myExchange", "");
         queue.subscribe(amqpMessageHandler);
   });
});
```

Socket.io – 服务器端

```
var express = require('express')
   , http = require('http')
   , amqp = require('amqp')
   ....;

server.listen(8081);
...
var amqpCon = amqp.createConnection(...);

io.sockets.on('connection', function (socket) {

  function amqpMessageHandler(message, headers, deliveryInfo) {
     var m = JSON.parse(message.data.toString());
     socket.emit('tick', m);
   };
   amqpCon.queue("", {},
      function(queue) {
         queue.bind("myExchange", "");
         queue.subscribe(amqpMessageHandler);
   });
});
```

处理 socket.io 连接

Socket.io – 服务器端

```
var express = require('express')
   , http = require('http')
   , amqp = require('amqp')
   ....;

server.listen(8081);
...
var amqpCon = amqp.createConnection(...);

io.sockets.on('connection', function (socket) {

  function amqpMessageHandler(message, headers, deliveryInfo) {
     var m = JSON.parse(message.data.toString());
     socket.emit('tick', m);
   };
   amqpCon.queue("", {},
      function(queue) {
         queue.bind("myExchange", "");
         queue.subscribe(amqpMessageHandler);
   });
});
```

提交到 AMQP 队列

Socket.io – 服务器端

```
var express = require('express')
   , http = require('http')
   , amqp = require('amqp')
   ....;

server.listen(8081);
...
var amqpCon = amqp.createConnection(...);

io.sockets.on('connection', function (socket) {

  function amqpMessageHandler(message, headers, deliveryInfo) {
     var m = JSON.parse(message.data.toString());
     socket.emit('tick', m);
   };
   amqpCon.queue("", {},
      function(queue) {
         queue.bind("myExchange", "");
         queue.subscribe(amqpMessageHandler);
   });
});
```

重新发布为

Socket.io – 客户端

```
<html>
<body>

The event is <span data-bind="text: ticker"></span>

<script src="/socket.io/socket.io.js"></script>
<script src="/knockout-2.0.0.js"></script>
<script src="/clock.js"></script>

</body>
</html>
```

绑定到模型

clock.js

```
var socket = io.connect(location.hostname);

function ClockModel() {
  self.ticker = ko.observable(1);
  socket.on('tick', function (data) {
   self.ticker(data);
  });
};

ko.applyBindings(new ClockModel());
```

Socket.io – 客户端

```
<html>
<body>

The event is <span data-bind="text: ticker"></span>

<script src="/socket.io/socket.io.js"></script>
<script src="/knockout-2.0.0.js"></script>
<script src="/clock.js"></script>

</body>
</html>
```

连接到 socket.io

clock.js

```
var socket = io.connect(location.hostname);

function ClockModel() {
  self.ticker = ko.observable(1);
  socket.on('tick', function (data) {
   self.ticker(data);
  });
};

ko.applyBindings(new ClockModel());
```

Socket.io – 客户端

```
<html>
<body>

The event is <span data-bind="text: ticker"></span>

<script src="/socket.io/socket.io.js"></script>
<script src="/knockout-2.0.0.js"></script>
<script src="/clock.js"></script>

</body>
</html>
```

clock.js

```
var socket = io.connect(location.hostname);

function ClockModel() {
  self.ticker = ko.observable(1);
  socket.on('tick', function (data) {
   self.ticker(data);
  });
};

ko.applyBindings(new ClockModel());
```

订阅到 Tick 事

Socket.io – 客户端

```
<html>
<body>

The event is <span data-bind="text: ticker"></span>

<script src="/socket.io/socket.io.js"></script>
<script src="/knockout-2.0.0.js"></script>
<script src="/clock.js"></script>

</body>
</html>
```

clock.js

```
var socket = io.connect(location.hostname);

function ClockModel() {
  self.ticker = ko.observable(1);
  socket.on('tick', function (data) {
   self.ticker(data);
  });
};

ko.applyBindings(new ClockModel());
```

更新模型

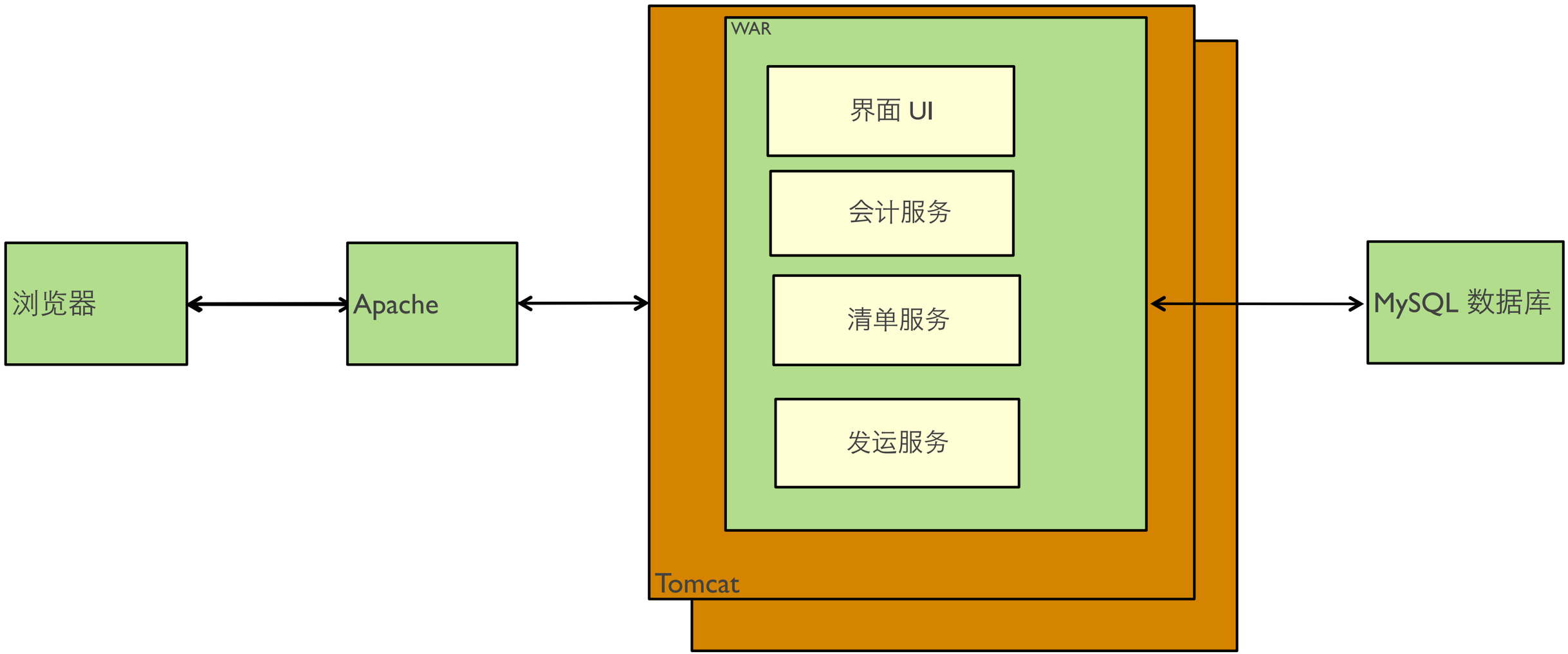
议题

- 整体性（有时不尽人意）
- 将应用程序分解为服务
- 服务如何通信？
- 表示层设计
- Cloud Foundry 有何帮助

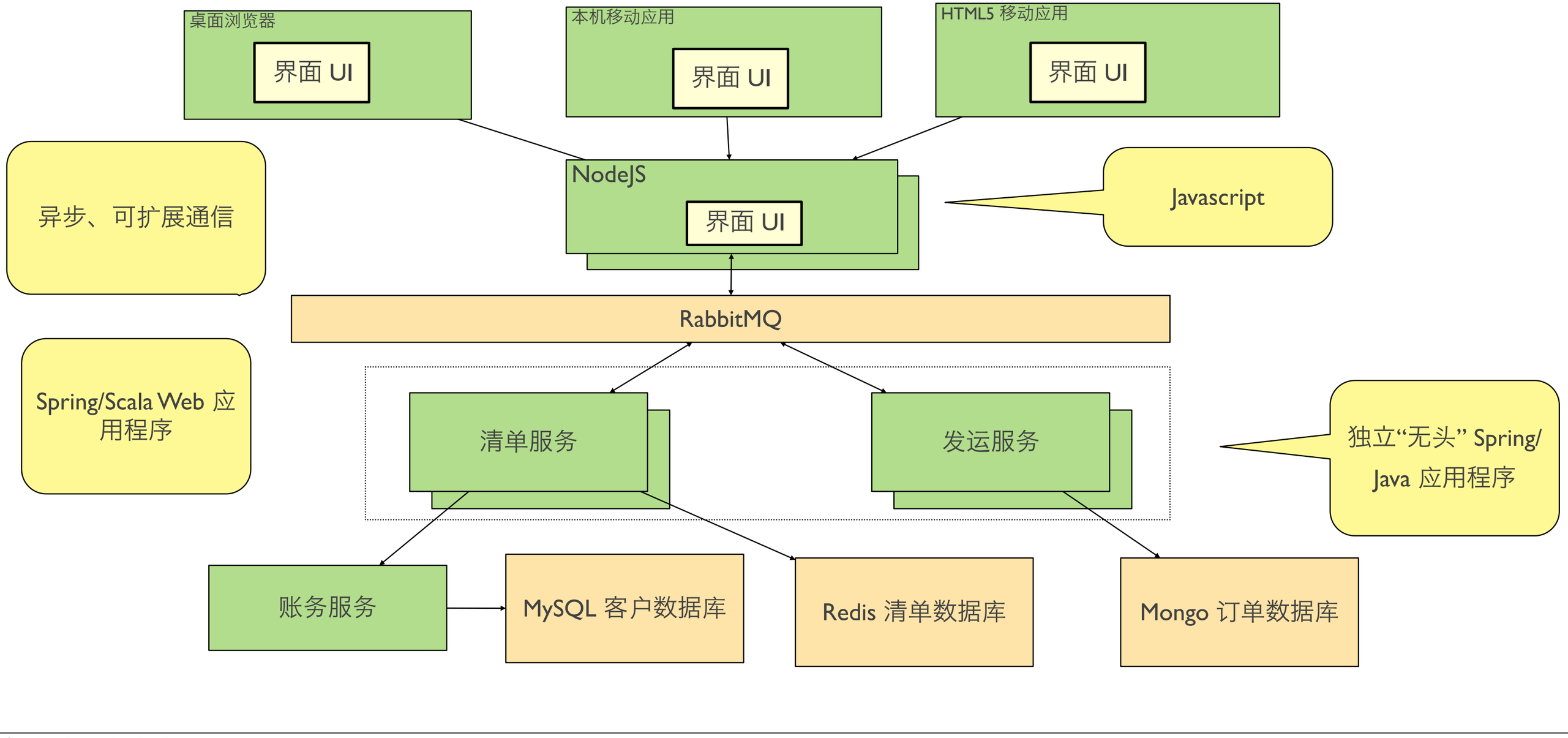
议题

- 整体性（有时不尽人意）
- 将应用程序分解为服务
- 服务如何通信？
- 表示层设计
- Cloud Foundry 有何帮助

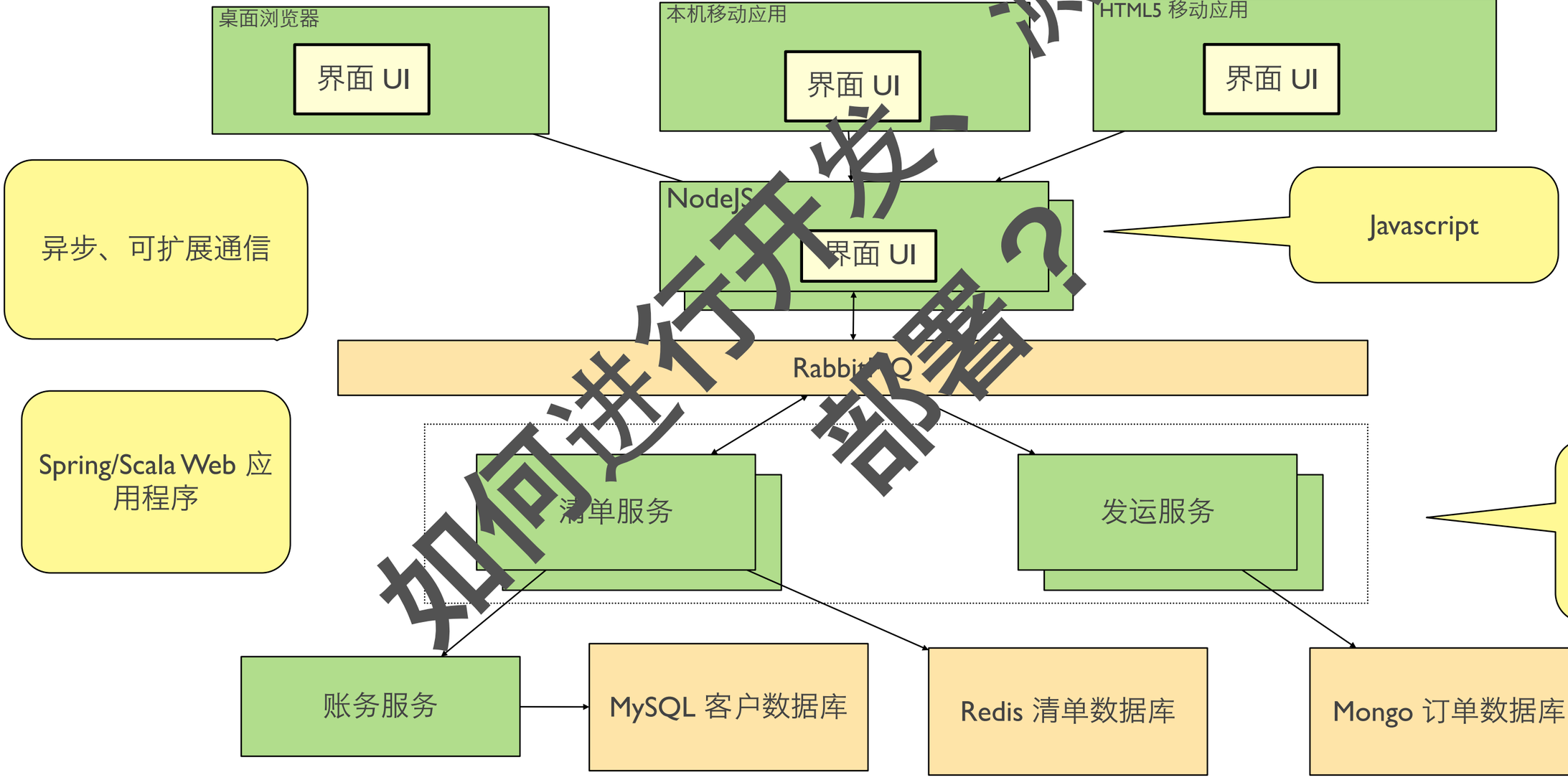
原始架构



更加先进的架构


桌面浏览器
界面 UI
本机移动应用
界面 UI
HTML5 移动应用
界面 UI
异步、可扩展通信
NodeJS
界面 UI
Javascript
RabbitMQ
Spring/Scala Web 应用程序
清单服务
发运服务
独立“无头” Spring/Java 应用程序
账务服务
MySQL 客户数据库
Redis 清单数据库
Mongo 订单数据库

更加先进的架构



传统工具：整体式应用程序

Web development that doesn't hurt

Ruby on Rails® is an open-source web framework that's optimized for programmer happiness and sustainable productivity. It lets you write beautiful code by favoring convention over configuration.

开发模块化的应用程序更加困难

开发模块化的应用程序更加困难

- 需要更多的移动部件来管理

开发模块化的应用程序更加困难

- 需要更多的移动部件来管理
  - 平台服务：SQL、NoSQL、RabbitMQ

开发模块化的应用程序更加困难

- 需要更多的移动部件来管理
  - 平台服务：SQL、NoSQL、RabbitMQ
  - **应用服务：您的代**码

开发模块化的应用程序更加困难

- 需要更多的移动部件来管理
  - 平台服务：SQL、NoSQL、RabbitMQ
  - **应用服务：您的代**码
- 谁来设置环境：

开发模块化的应用程序更加困难

- 需要更多的移动部件来管理
  - 平台服务：SQL、NoSQL、RabbitMQ
  - **应用服务：您的代**码
- 谁来设置环境：
  - 开发人员沙盒？

开发模块化的应用程序更加困难

- 需要更多的移动部件来管理
  - 平台服务：SQL、NoSQL、RabbitMQ
  - **应用服务：您的代**码
- 谁来设置环境：
  - 开发人员沙盒？
  - ...

开发模块化的应用程序更加困难

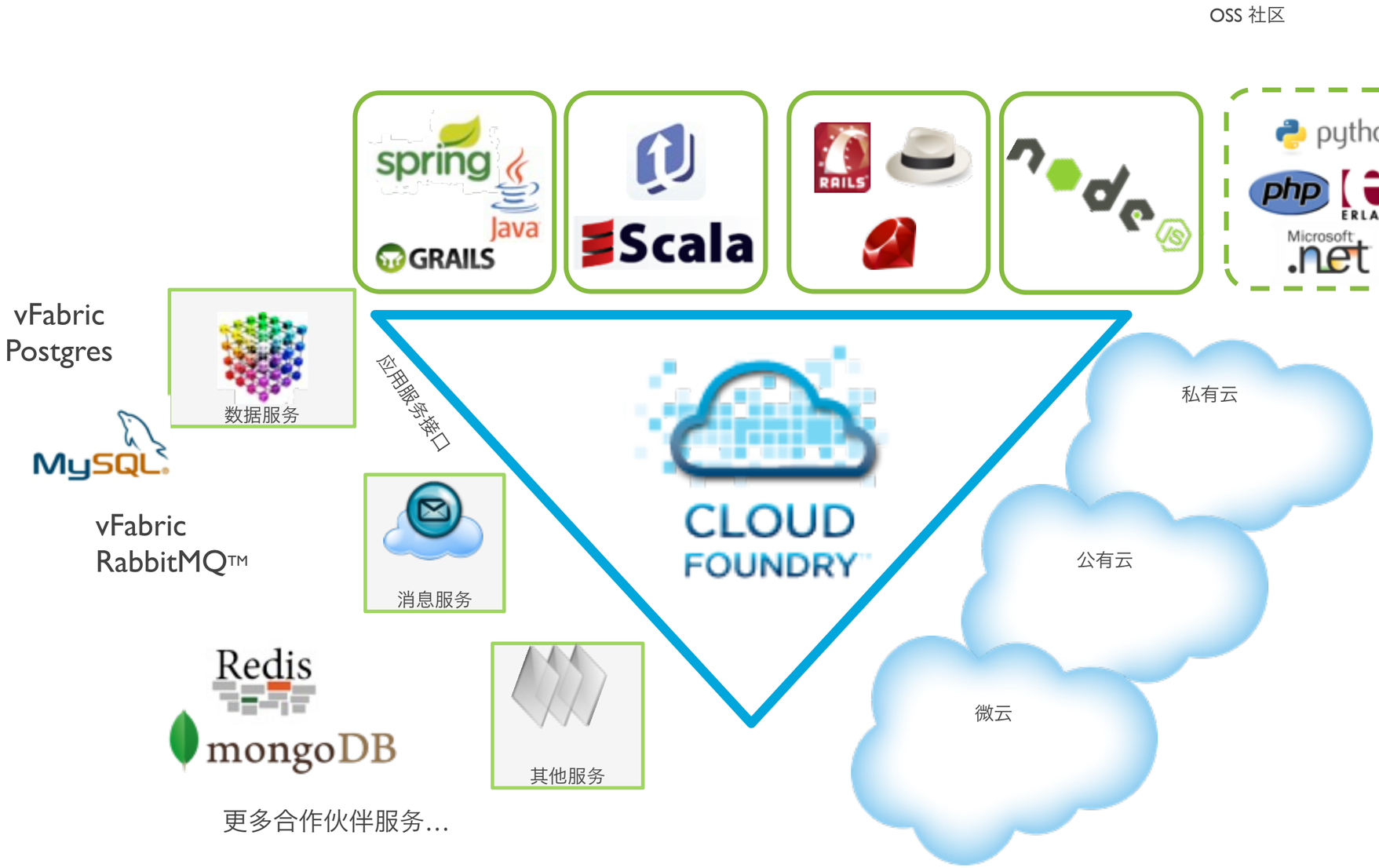
- 需要更多的移动部件来管理
  - 平台服务：SQL、NoSQL、RabbitMQ
  - **应用服务：您的代**码
- 谁来设置环境：
  - 开发人员沙盒？
  - ...
  - QA 环境？

开发模块化的应用程序更加困难

- 需要更多的移动部件来管理
  - 平台服务：SQL、NoSQL、RabbitMQ
  - **应用服务：您的代**码
- 谁来设置环境：
  - 开发人员沙盒？
  - ...
  - QA 环境？

Cloud Foundry 自有高招...

轻松的多语言应用开发和服务配置



创建一个平台服务实例

```
$ vmc create-service mysql --name mysql1
Creating Service: OK

$ vmc services
......

=========== Provisioned Services ============

+--------------+----------+
| Name         | Service  |
+--------------+----------+
| mysql1       | mysql    |
+--------------+----------+
```

多应用程序清单 – 第 1 部分

```
---
applications:
 inventory/target:
   name: inventory
   url: cer-inventory.chrisr.cloudfoundry.me
   framework:
     name: spring
     info:
       mem: 512M
       description: Java SpringSource Spring Application
       exec:
   mem: 512M
   instances: 1
   services:
     si-rabbit:
       type: :rabbitmq
     si-mongo:
       type: :mongodb
     si-redis:
       type: :redis
```

应用程序的路径

所需的平台服务

多应用程序清单 – 第 2 部分

```
store/target:
  name: store
  url: cer-store.chrisr.cloudfoundry.me
  framework:
    name: spring
    info:
      mem: 512M
      description: Java SpringSource Spring Application
      exec:
mem: 512M
instances: 1
services:
  si-mongo:
    type: :mongodb
  si-rabbit:
    type: :rabbitmq
```

应用程序的路径

所需的平台服务

一条命令即可创建平台服务并部署应用程序

- **$ vmc push**
- Would you like to deploy from the current directory? [Yn]:
- Pushing application 'inventory'...
- Creating Application: OK
- Creating Service [si-rabbit]: OK
- Binding Service [si-rabbit]: OK
- Creating Service [si-mongo]: OK
- Binding Service [si-mongo]: OK
- Creating Service [si-redis]: OK
- Binding Service [si-redis]: OK
- Uploading Application:
- Checking for available resources: OK
- Processing resources: OK
- Packing application: OK
- Uploading (12K): OK
- Push Status: OK
- Staging Application 'inventory': OK
- Starting Application 'inventory': OK
- Pushing application 'store'...
- Creating Application: OK
- Binding Service [si-mongo]: OK
- Binding Service [si-rabbit]: OK
- Uploading Application:
- Checking for available resources: OK
- Processing resources: OK
- Packing application: OK

vmc push：

- 读取清单文件
- 创建所需的平台服务
- 部署所有应用程序

Micro Cloud Foundry：开发人员的新沙盒

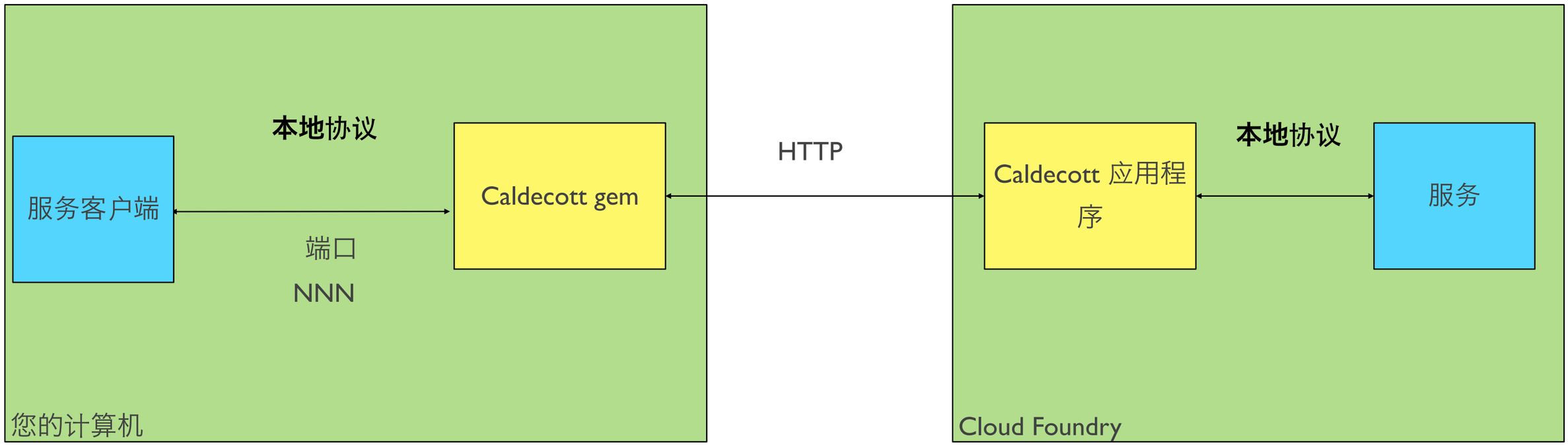
一个打包为 VMware 虚拟机的 PaaS

用作开发人员沙盒

- 使用来自 Junit 集成测试的服务
- 开发自己的应用程序以进行功能测试
- 从 STS 进行远程调试

使用 Caldecott 建立通往服务的隧道

Caldecott = HTTP之上的 TCP



使用 Caldecott…

```
$ vmc tunnel
1: mysql-135e0
2: mysql1
Which service to tunnel to?: 2
Password: ********
Stopping Application: OK
Redeploying tunnel application 'caldecott'.
Uploading Application:
  Checking for available resources: OK
  Packing application: OK
  Uploading (1K): OK
Push Status: OK
Binding Service [mysql1]: OK
Staging Application: OK
Starting Application: OK
Getting tunnel connection info: OK

Service connection info:
  username : uMe6Apgw00AhS
  password : pKcD76PcZR7GZ
  name     : d7cb8afb52f084f3d9bdc269e7d99ab50

Starting tunnel to mysql1 on port 10000.
1: none
2: mysql
Which client would you like to start?: 2
```

…使用 Caldecott

```
Launching 'mysql --protocol=TCP --host=localhost --port=10000 --user=uMe6Apgw00AhS --
    password=pKcD76PcZR7GZ d7cb8afb52f084f3d9bdc269e7d99ab50'


Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 10944342
Server version: 5.1.54-rel12.5 Percona Server with XtraDB (GPL), Release 12.5, Revision 188

Copyright (c) 2000, 2011, Oracle and/or its affiliates. All rights reserved.

Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

mysql>
```

借助 Caldecott 运行 JUnit 测试

配置您的测试代码，以使用端口 + 连接信息

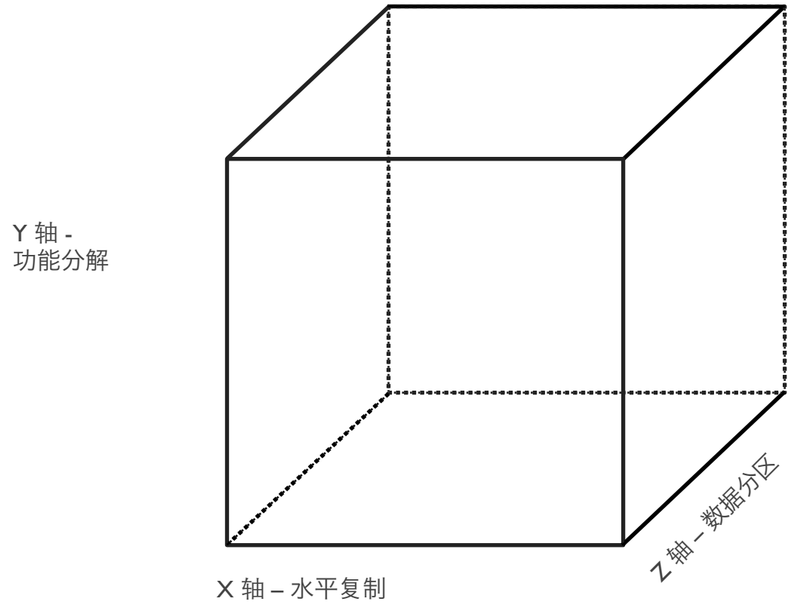
```
Service connection info:
  username : uFZpMHcVgMyjN
  password : pMYfxETX3dcxA
  name     : da285916ae4234b91a6ceadb638aa8365

Starting tunnel to survey-mysql on port 10000.
```

总结

整体式应用程序

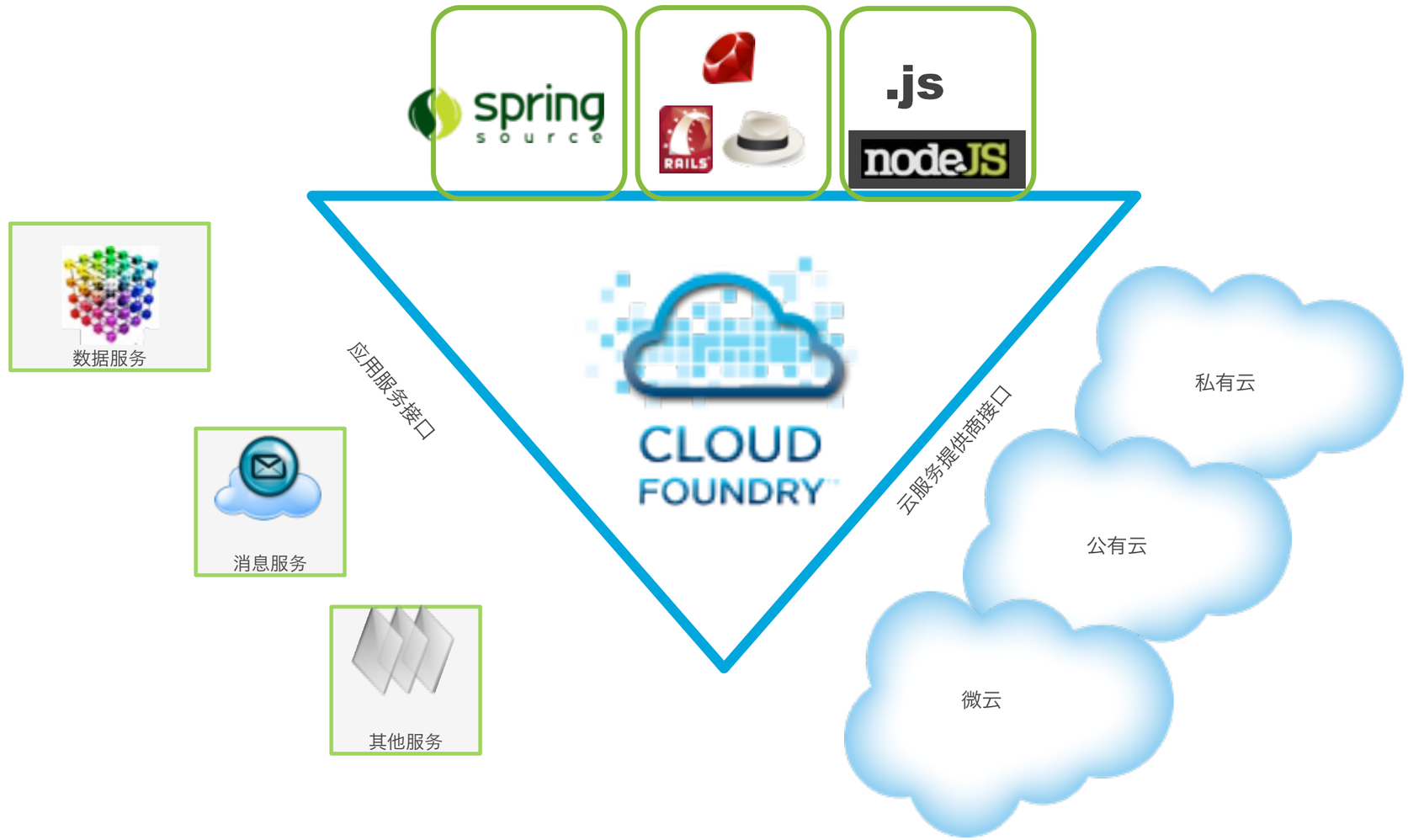
易于开发和部署

应用扩展立方体



- 模块化、多语言的可扩展应用程序
- 独立开发、部署和扩展服务

Cloud Foundry 有何帮助



@crichardson crichardson@vmware.com
http://plainoldobjects.com/presentations/

问题？

www.cloudfoundry.com @cloudfoundry

Cloud Foundry 启动营

在www.cloudfoundry.com注册账号并成功上传应用程序,

即可于12月8日中午后凭账号ID和应用URL到签到处换取Cloud Foundry主题卫衣一件。

iPhone5 等你拿

第二天大会结束前， 请不要提前离开， 将填写完整的意见反馈表投到签到处的抽奖箱内，
即可参与“iPhone5”抽奖活动。

Birds of a Feather 专家面对面

所有讲师都会在课程结束后，到紫兰厅与来宾讨论课程上的问题