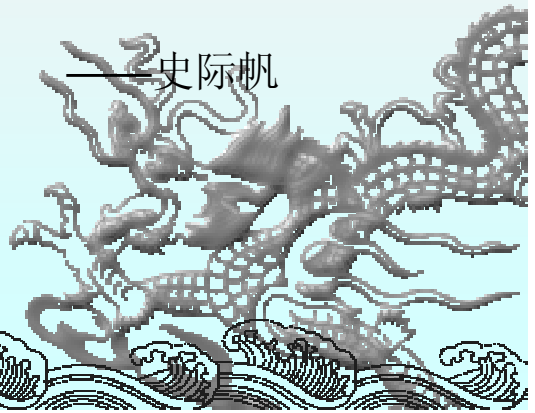
# Framework for Integrated Test: Beauty Through Fragility

——*Michael Feathers*

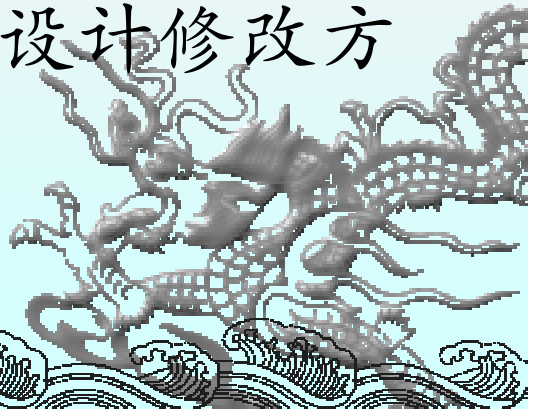
——《代码之美》第六章

——史际帆



# 作者简介： *Michael Feathers*

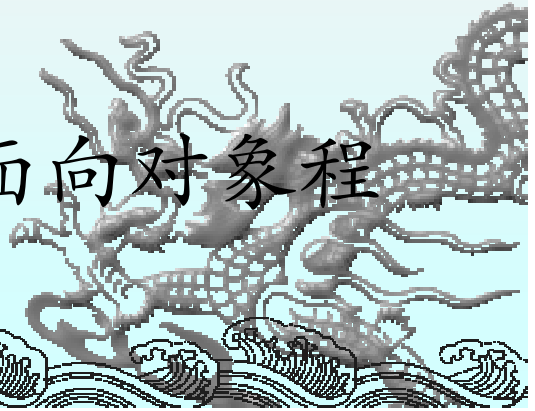
- Object Mentor公司顾问
- 主要工作是与世界各地不同的团队合作、培训以及指导。
- 一直活跃于Agile社群。开发了CppUnit（及将Junit移植到C++），FitCpp（及将Fit移植到C++）。
- 编写了《Working Effective with Legacy Code》
- 大多数时间用在研究大型代码库的设计修改方面。



# 好的设计

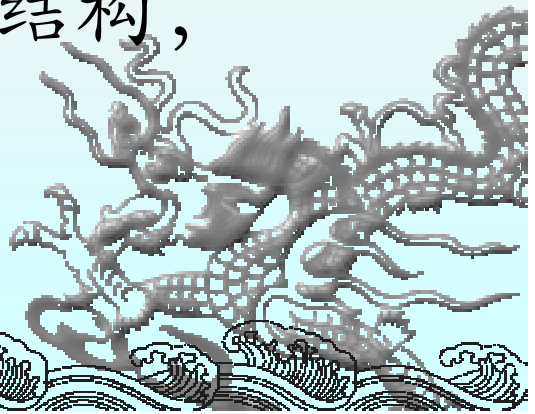
- 不是遵规蹈距：
  - 用公有变量时想到起糟糕症状而不敢用
  - 实现继承时考虑尽可能用委托而不是继承
- 不是显现程序多么复杂

- 而是具体问题具体分析，运用最简洁的代码实现高效的程序
- FIT就另辟蹊径，打破常规，在面向对象程序设计中表现出另类的美！



# 框架

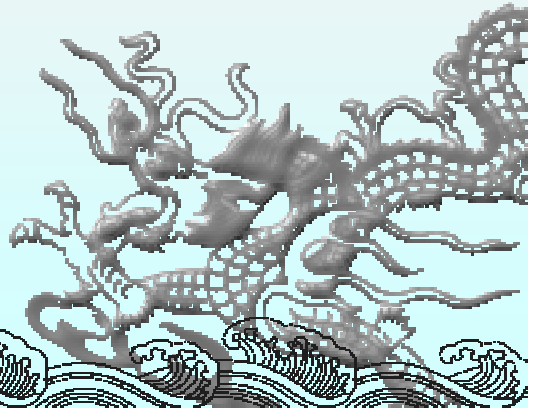
- 框架：框架是一个应用程序的半成品。框架提供了可以在应用程序之外共享可复用的公共结构。开发者把框架融入他们自己的应用程序，并加以扩展，以满足他们特定的需要。框架和工具包的不同之处在于，框架提供了一致的结构，而不仅仅是一组工具类。



# 单元测试

- 单元测试：单元测试测的是独立的一个工作单元。在Java应用程序中，“独立的一个工作单元”常常指的是一个方法（但不总是如此）。作为对比，集成测试和接收测试则检查多个组件如何交互。一个工作单元是一项任务，它不依赖于其他任何任务的完成。

① 单元测试可以降低不确定性从而降低风险
② 单元测试可以帮助优化设计
③ 单元测试用例可以当文档使用



# FIT测试框架

- 利用HTML表格来编写执行测试
- 对页面上的每一张表格创建一个fixture对象，读取表格单元中的值，检查应用程序返回值与表格中的期望值否相符，如果不符，则将单元格设为红色，反之设为绿色。

| **MarketEvaluation** | | | | |
|---|---|---|---|---|
| Symbol | Shares | Price | Portfolio | Attempt Fill? |
| PLI | 1100 | 1560 | no | no |

| **MarketEvaluation** | | | | |
|---|---|---|---|---|
| Symbol | Shares | Price | Portfolio | Attempt Fill? |
| PLI | 1100 | 1560 | no | no expected |
| | | | | yes actual |

# FIT测试框架

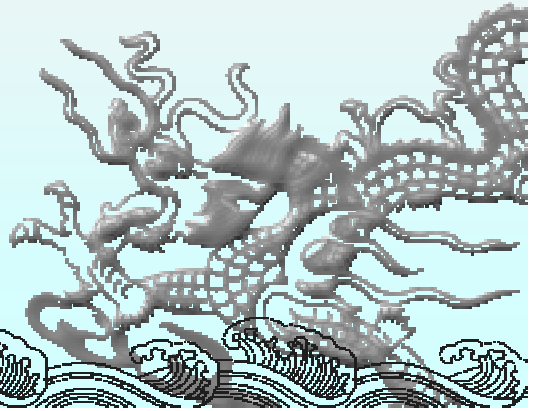
- 但是在《集成测试框架——用FIT进行敏捷软件测试》中要求表格命名Calculate***,才可以被测试系统进行自动化测试。
- 如：

| CalculateDiscount | |
|---|---|
| amount | discount() |
| 0.00 | 0.00 |
| 100.00 | 0.00 |
| 999.00 | 0.00 |
| 1000.00 | 0.00 expected<br>50.0 actual |
| 1010.00 | 50.50 |
| 1200.00 | 60.00 |
| 2000.00 | 100.00 |

# FIT测试框架

- 报告：红绿灯
  - 绿色：测试通过
  - 红色：测试因为异常而失败，可能还会提供一些补充信息
  - 黄色：测试的部分没有完全实现，或者出错。
  - 灰色：表格的部分因为某些已知原因没有执行，也许因为测试没有完全实现。

JUnit中名言：Keep the bar green to keep the code clean.

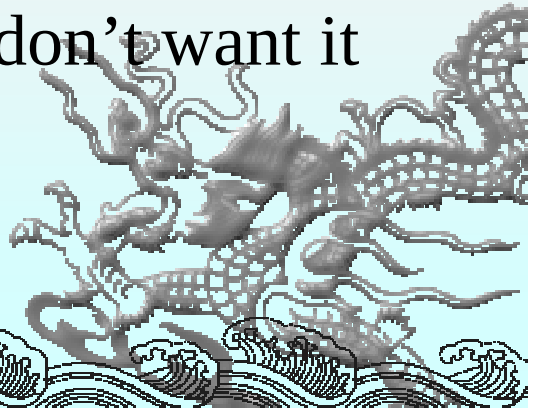


# FIT测试框架

- FIT的关键特性就是可以测试文档，这只需将这些带表格的文档用HTML语言写，或用Word写后存为HTML格式，或用支持“保存为HTML文件”的编辑器都可以。
- FIT的工作描述完了，大家可以猜测一下，完成这样的工作需要多长的代码呢?
- 实际上，FIT使用的类少得惊人，在展现FIT之前，我们先看看一般如何设计“应用程序接口”的。

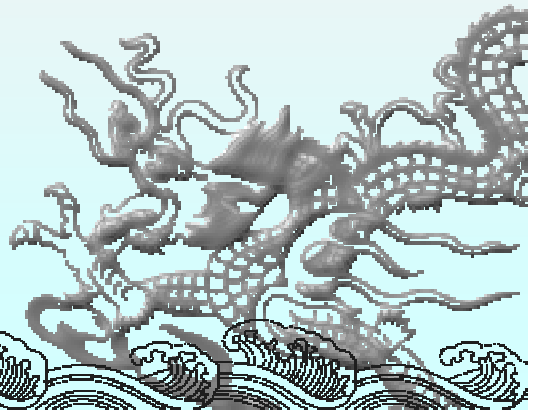
# 一般设计API（Application Programming Interface）原则

- 1. Be very careful about what you make public: control visibility to keep the published parts of the framework small.
  - 尽量减少框架公开部分
- 2. Use interfaces.
  - 使用接口
- 3. Provide well-defined “hook points” that permit extensibility in the places where you intend it to occur.
  - 可扩展的点上提供良好“挂钩点”
- 4. Prevent extension in places where you don’t want it to occur.
  - 不希望被扩展处禁用扩展点



# 一个一般API的例子——JavaMail

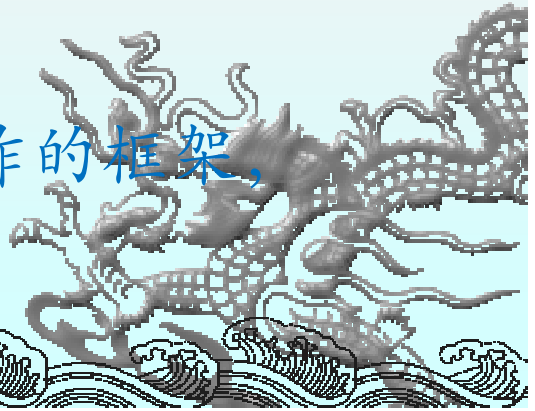
```
package javax.mail;
public final class Session
{
    ...
    public static Session getDefaultInstance(Properties props);
    ...
    public Store getStore( ) throws NoSuchProviderException;
...
}
public abstract class Store extends javax.mail.Service {
    ...
    public void addStoreListener(StoreListener listener);
    ...
}
```



# 一个一般API的例子——JavaMail

- 收取邮件必先取得一个Session（会话）对象的引用：
  - final类不能创建其派生类（禁止扩展）
  - 而Session没有构造函数，只能使用getDefaultInstance方法
- 有Session对象后，使用getStore获取一个包含邮件文件对象的Store对象，JavaMail提供了Store的一些派生类，故返回的Store对象十多种派生类对象中的一种（受管制的扩展点）
  - 而Store对象上挂有一个监听器类，在Store对象被改动时收到通知（“挂钩点”）

这是一个不错的框架自我保护！
但按照这种原则，一个与FIT完成同样工作的框架，
至少需要比FIT多三四倍的类才可以实现！
所以FIT必然打破了这种做法！



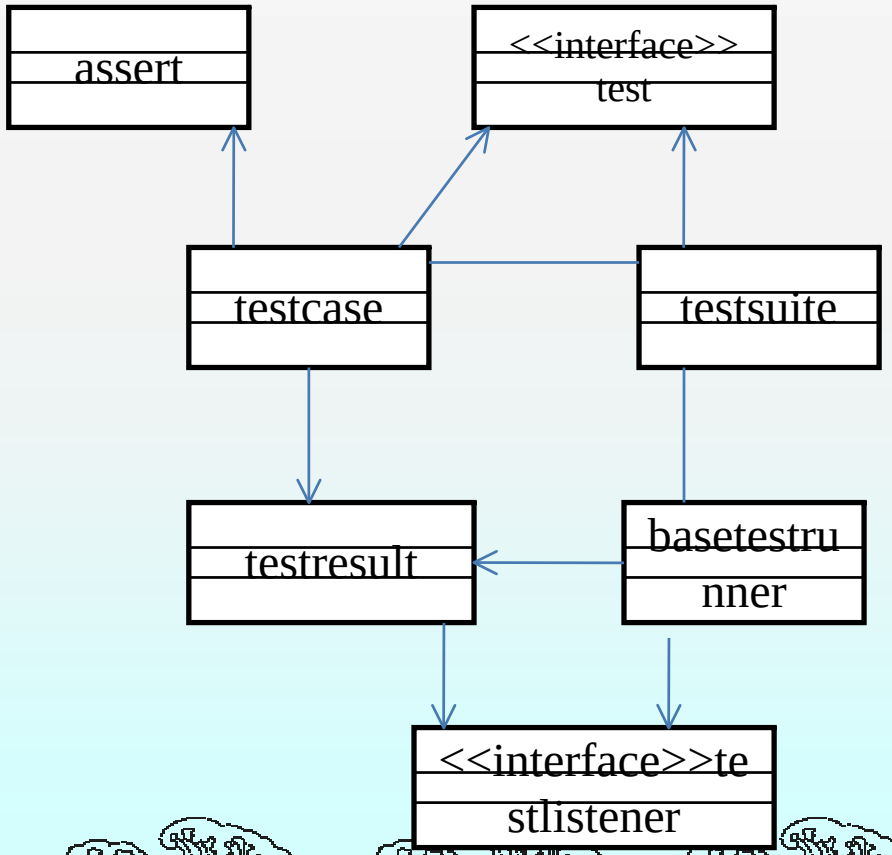
# JUnit的核心

- 七个核心类及接口：

| 类/接口 | 作用 |
| --- | --- |
| Assert | 但条件成立时assert方法保持沉默，但若条件不成立就抛出异常 |
| TestResult | TestResult包含了测试中发生的所有错误或者失败 |
| Test | 可以运行Test并把结果传递给TestResult |
| TestListener | 测试中若产生事件（开始、结束、错误、失败）会通知TestListener |
| TestCase | TestCase定义了可以运行与多项测试的环境（或者说说固定设备） |
| TestSuite | TestSuite运行一组TestCase（他们可能包含其他TestSuite），它是Test的组合 |
| BaseTestRunner | TestRunner是用来启动测试的用户界面，BaseTestRunner是所有TestRunner的超类 |

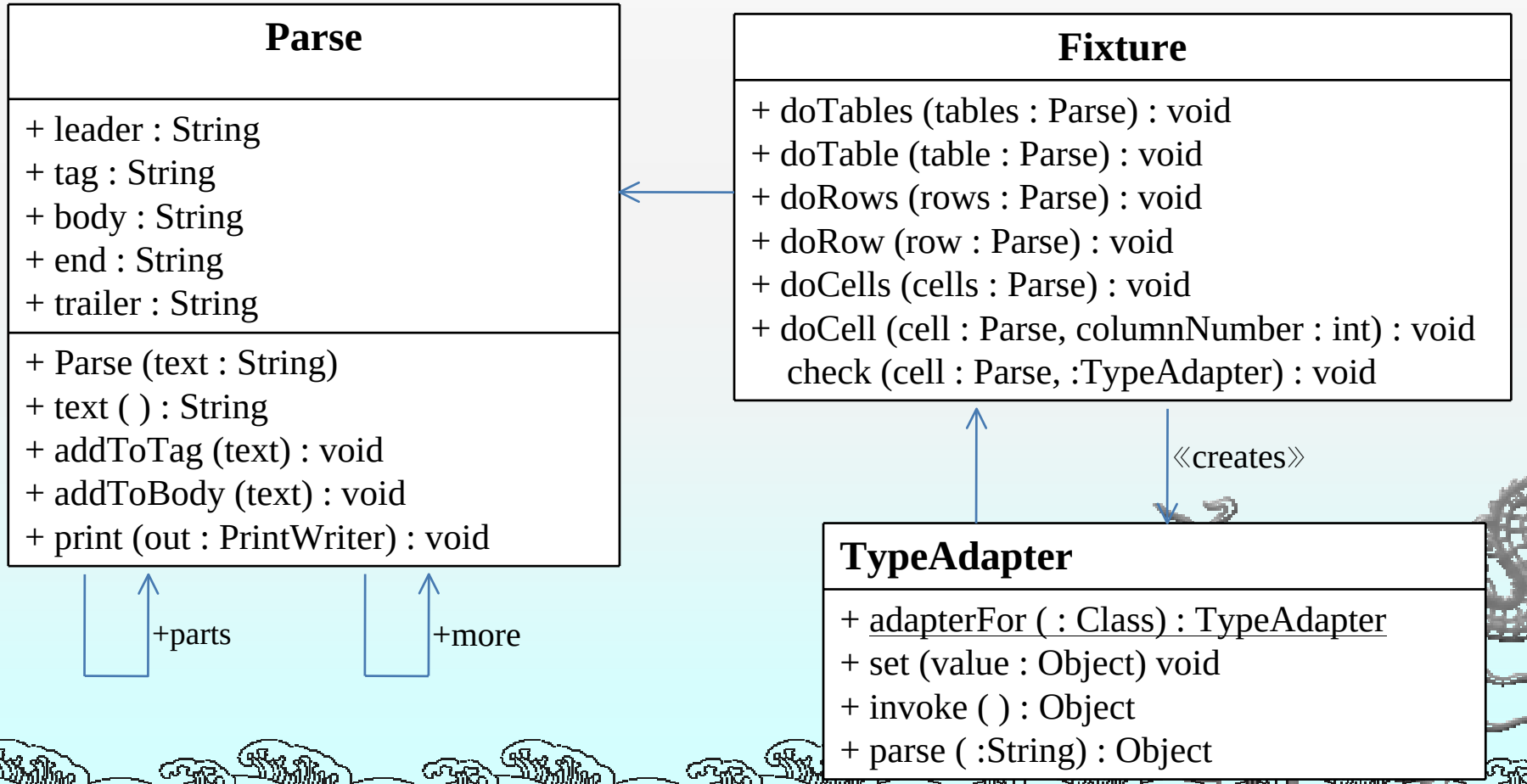
# JUnit的核心

- 而TestCase 有10个基本方法……
- 整体较复杂



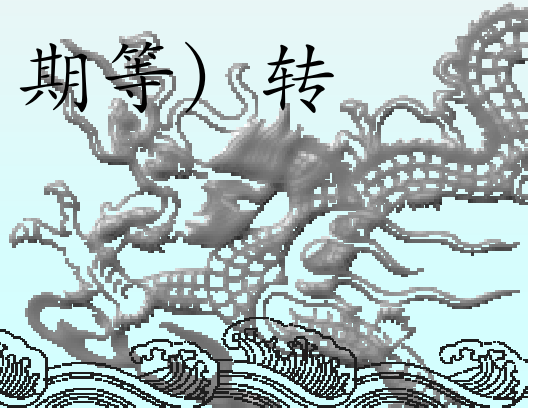
# FIT测试框架

- FIT的核心类只有三个



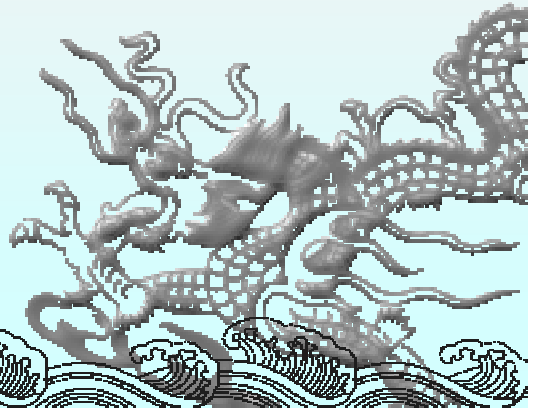
# FIT测试框架

- Parse：其中的构造函数将接受到的字符串递归构造出一棵由Parse对象构成的Parse树，树中节点之间通过parts和more两个成员函数来联系，而其中的每个Parse对象代表着文档的某个部分：每张表格、表格每一行、每一行的每个单元格。
- Fixture：遍历这棵Parse树，并与要进行测试的应用交互，确定将单元格涂成红色还是绿色。其有许多派生类，进行不同的操作，自己也可以定义。
- TypeAdapter：将测试值（数字、日期等）转换为文本或反之转换回来。



# FIT的开放式框架

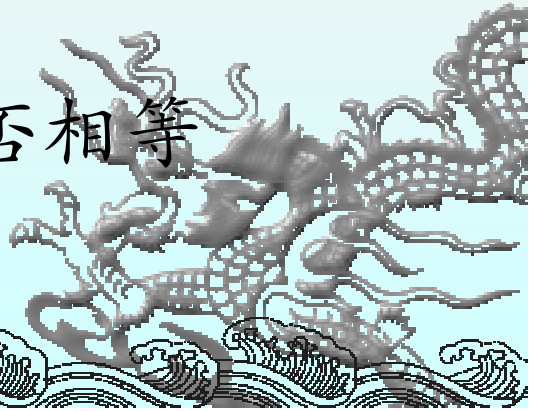
- FIT的框架与前面所述的完全不同，FIT中的大部分成分是公有的，甚至成员数据。
- FIT的框架不像传统的框架那样只提供一些设计好的扩展点，他将全部框架完全开放！



# FIT的开放式框架——TypeAdapter类

管理单元格文本与Java值得转换：

- get()，从应用程序中一个对象的特定域中获取数据
- set()，往一对象的特定域中写入数据
- invoke()，调用对象的特定方法（不含参数）
- toString，提供数值的String形式，例如用于表格
- parse()，从单元格中解析文本并转换成合适的数据类型
- equals()，检查两个数据类型的值是否相等



# FIT的开放式框架——Parse类

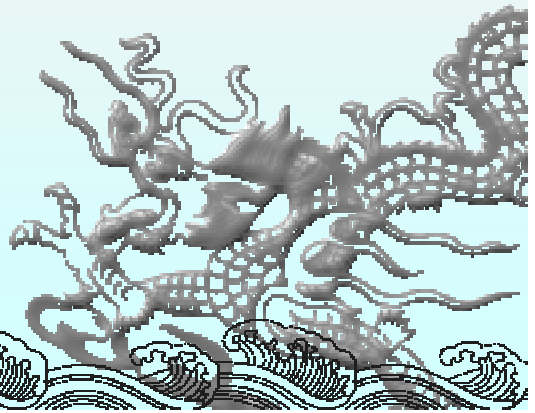
- FIT中的解析工作全部由Parse类完成，它奇迹般的构造函数负责构造出一整棵树。

- 每个Parse实例有5个公有成员，2个Parse引用：

```
public String leader;
public String tag;
public String body;
public String end;
public String trailer;

public Parse more;
public Parse parts;
```

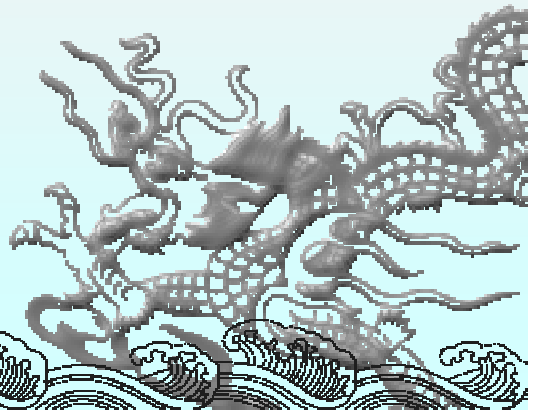
- 调用Parse：

```
String input = read(new File(argv[0]);
Parse parse = new Parse(input);
```

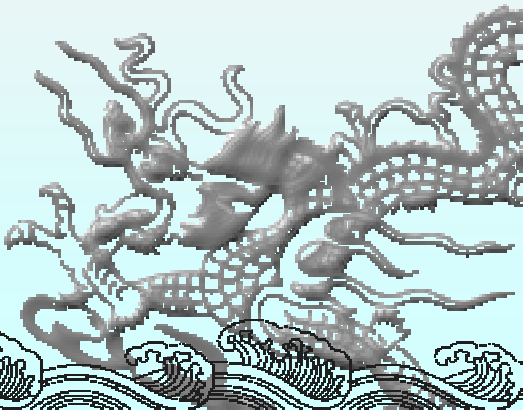


# FIT的开放式框架——Parse类

```
static String tags[] = {"table", "tr", "td"};

public Parse (String text) throws ParseException {
    this (text, tags, 0, 0);
}
public Parse (String text, String tags[]) throws ParseException {
    this (text, tags, 0, 0);
}

public Parse (String text, String tags[], int level, int offset) throws ParseException
{
    String lc = text.toLowerCase( );
    int startTag = lc.indexOf("<"+tags[level]);
    int endTag = lc.indexOf(">", startTag) + 1;
    int startEnd = lc.indexOf("</"+tags[level], endTag);
    int endEnd = lc.indexOf(">", startEnd) + 1;
    int startMore = lc.indexOf("<"+tags[level], endEnd);
```



```
if (startTag<0 || endTag<0 || startEnd<0 || endEnd<0) {
        throw new ParseException ("Can't find tag: "+tags[level], offset);
}
leader = text.substring(0,startTag);
tag = text.substring(startTag, endTag);
body = text.substring(endTag, startEnd);
end = text.substring(startEnd,endEnd);
trailer = text.substring(endEnd);

if (level+1 < tags.length) {
        parts = new Parse (body, tags, level+1, offset+endTag);
        body = null;
}

if (startMore>=0) {
        more = new Parse (trailer, tags, level, offset+endEnd);
        trailer = null;
}
}
```



## http://www.pku.edu.cn/schools/yxsz.jsp 北大院系设置HTML原文件（节选）

```
<table width="85%" class="t6" valign="top">
  <tr><th colspan="4" >理学部</th>
    </tr>
    <tr>
    <td width="23%" align="left"  ><a href="http://www.math.pku.edu.cn/">数学科学学院</a></td>
    <td width="27%" align="left"  ><a href="http://www.phy.pku.edu.cn/">物理学院</a></td>
    <td width="25%" align="left"  ><a href="http://www.chem.pku.edu.cn/">化学与分子工程学院
    </a></td>
    <td width="25%" align="left"  ><a href="http://www.bio.pku.edu.cn/">生命科学学院</a></td>
  </tr>
……
<tr>
    <td align="left"  ><a href="http://stl.szpku.edu.cn/">国际法学院</a></td>
    <td align="left"  ><a href="http://rw.szpku.edu.cn/info/">人文社会科学学院</a></td>
    <td align="left"  ><a href="http://mprc.szpku.edu.cn/">微处理器研究开发中心</a></td>
    <td align="left"  ><a href="http://netlab.szpku.edu.cn/">互联网信息工程研发中心</a></td>
  </tr>

</table>
```

# FIT的开放式框架——Parse类

- 构造成树之后，最顶层的一个Parse对象实际上就代表了整个HTML文档，可以调用print方法打印整个文本:

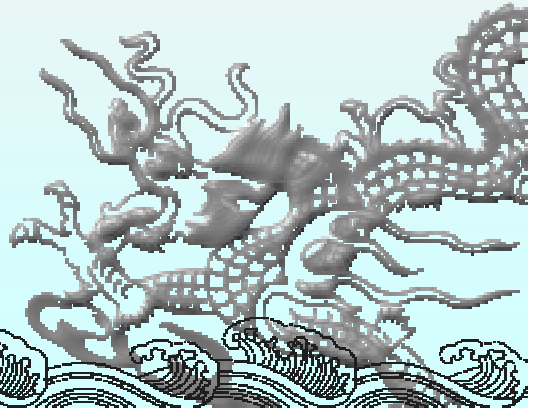
```
public void print(PrintWriter out) {
    out.print(leader);
    out.print(tag);
    if (parts != null) {
        parts.print(out);
    } else {
        out.print(body);
    }
    out.print(end);
    if (more != null) {
        more.print(out);
    } else {
        out.print(trailer);
    }
}
```

- 整个Parse的实现完全突破了传统:

1.构造函数做了很多事情

2.优美对称地将文本分为几百个Parse对象

3.一个简单的print又将他们重建起来



# FIT的开放式框架——Fixture类

每个表格调用dotable:

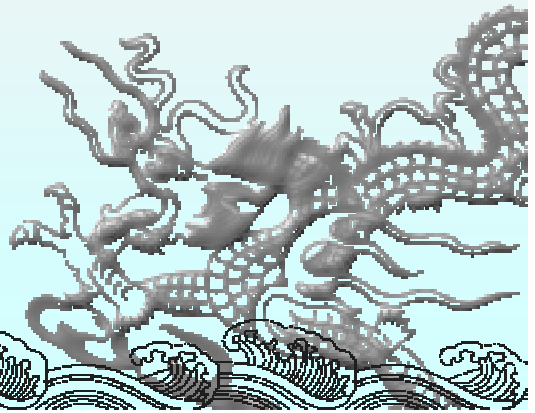
```
public void doTable(Parse table) {
    doRows(table.parts.more);
}
```

再调用doRows

```
public void doRows(Parse rows) {
    while (rows != null) {
        doRow(rows);
        rows = rows.more;
    }
}
```

再调用doRow

```
public void doRow(Parse row) {
    doCells(row.parts);
}
```



# FIT的开放式框架——Fixture类

再调用doCells

```
public void doCells(Parse cells) {
        while (cells != null) {
                doCell(cells, columnNumber);
                cells = cells.more;
        }
}
```

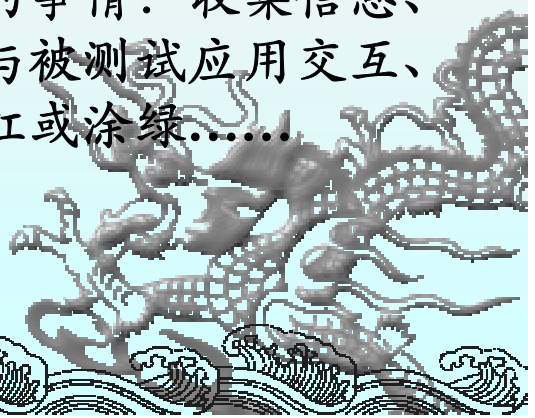
再调用doCell

```
public void doCell(Parse cell, int columnNumber) {
        ignore(cell);
}
```

其中ignore方法将单元格涂成灰色：

```
public static void ignore (Parse cell) {
        cell.addToTag(" bgcolor=\"#efefef\"");
        ignores++;
}
```

可以重写每个函数，然他们做我们想让他做的事情：收集信息、保存信息、与被测试应用交互、将单元格涂红或涂绿……

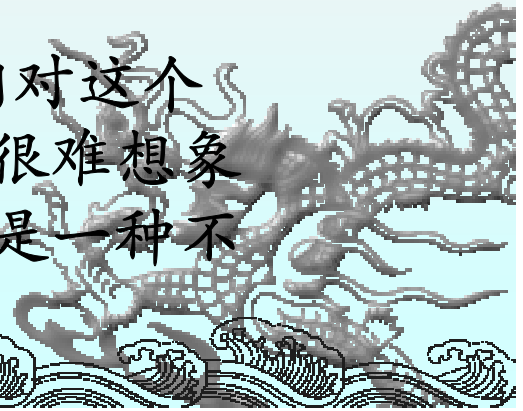


# FIT的开放式框架——Fixture类

- Fixture的代码也与传统格格不入:

1. 用户不是将代码插入框架，而是从一个类派生，重写其某些缺省动作。
2. 框架没有任何保护机制，用户只要调用dotables方法，全部代码就公开了。
3. 公开的代码永远不能再动，一动就必然破坏客户代码。FIT的代价就是未来任何版本都不能收回用户对这些成员变量的访问权。

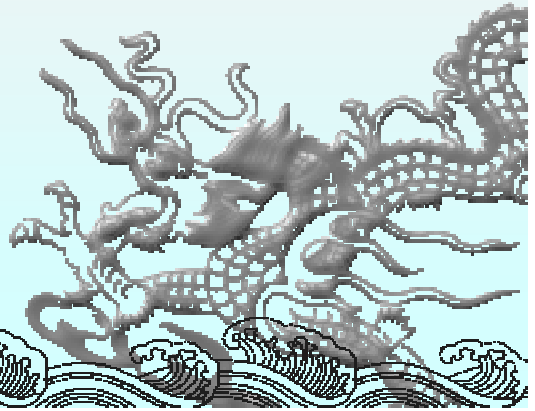
从经典角度，**Fixture**似乎很糟，可是如果我们对这个遍历很有信心，而且这些环节都是很稳定的，很难想象他有什么问题。而在特定情况下，这种决策也是一种不错的选择!



# 结论

- 传统：
  - 通过限制选择来保留可扩展性。
  - 使用一大堆类实现复杂功能，而每一个类保持独立封装。
- FIT的选择：
  - 框架灵活、简洁，方法设为公有，框架完全公开
  - 小心把握每一个类得内部分解

开放式风格，有时也是不错的选择!



# 谢谢！