《软件测试课程设计》讲稿（第 8 章）

第 10 周 · 2 学时（90 分钟）

广东金融学院 计算机学院 | 周宇文

讲稿可照读版 | 由课件逐页撰写

第 8 章　软件本地化测试

第 10 周 · 2 学时（90 分钟） | 教材：朱少民《软件测试方法和技术（第 4 版）》

《软件测试课程设计》第 8 章 讲稿（第 10 周 · 2 学时 90 分钟）

> 课程：软件测试课程设计（154442008） | 广东金融学院 计算机学院 | 专业选修
>
> 章节：第 8 章 软件本地化测试 —— 同一套代码，能不能在另一个语言和地区"像本地产品一样好用"
>
> 学时：**2 学时（90 分钟）**
>
> 教材：朱少民《软件测试方法和技术（第 4 版）》第 2 篇 软件测试技术
>
> 配套课件：逐页还原版 **46 页**（`lecture8/`，云端 http://111.231.0.226:2345/lecture8/index.html）
>
> 教案对应：第 10 次课「软件本地化测试」
>
> 本讲重点：国际化（i18n）与本地化（l10n）的区别；本地化测试的测试项（翻译质量、界面布局与截断、日期时间与货币格式、排序与检索、输入法与字符集、图片图标、法规与习俗）；测试组织（单语言/多语言环境、伪本地化、翻译验证与回归）
>
> 本讲难点：为什么"能显示中文"不等于"本地化做对了"；伪本地化如何在不翻译的情况下提前暴露布局问题；字符集与编码问题的定位思路

> **本讲稿为可照读完整版**：每页含【口播】【板书】【提问＋预设回答】【易错点/考点】【过渡】；带 ★ 的页为可压缩页。

一、教学目标

维度	目标（课后可检验）
知识	① 能区分国际化与本地化，并说明各自发生在哪个阶段、由谁负责；② 能列出本地化测试至少 8 类测试项；③ 能解释伪本地化的原理与价值；④ 能说出编码/字符集问题的常见现象与排查方向；⑤ 能说明本地化回归测试的范围怎么定
能力	① 能为一个已上线的中文系统列出"出海到英文与阿拉伯语市场"的本地化测试清单；② 能设计一套伪本地化验证方案；③ 能判断一个界面问题是翻译问题、布局问题还是编码问题
素质（思政）	① 由编码/乱码问题体会"细节决定成败"；② 由习俗与法规禁忌体会尊重文化差异与合规意识；③ 由翻译质量体会对用户负责；④ 由测试组织体会流程与严谨

本讲要建立的一个总印象：本地化测试测的不是"翻译对不对"，而是"**这个产品在这个市场里，像不像一个本地产品**"——语言、格式、习惯、法规，四样都要对。

二、时间分配总表

> **容量提示**：本章课件 **46 页**，讲稿是素材全量版（口播约 2 万字，≈85 分钟），课时 90 分钟，**基本可以按页讲完**；仍建议把纯图示页与工具页快速带过。

时段	页码	内容	分钟	备注
第 1 节	8.1–8.12	开场、i18n 与 l10n、本地化测试的定位与测试项	26	**测试项清单必讲**

第 1 节	8.13–8.26	界面布局与截断、格式本地化（日期/时间/货币/数字）、排序与检索	24	现场找截断例子
——	——	**课间休息 10 分钟**	——	——
第 2 节	8.27–8.38	字符集与编码、输入法、图片图标、法规与习俗	20	**编码问题必讲**
第 2 节	8.39–8.46	测试组织与执行、伪本地化、回归、小结与作业	15	伪本地化当堂设计

> **压缩优先级**：先压缩工具截图页与流程图示页 → 再压缩具体语种细节 → 最后才压缩"测试项清单""编码问题""伪本地化"三处。

三、逐页讲稿

> 每页第一段是可直接朗读的口播；其后是板书、提问与预设回答、易错点/考点、过渡语。
>
> 页码锚点与课件页一一对应（8.1–8.46）：http://111.231.0.226:2345/lecture8/index.html

【8.1】软件测试方法和技术（1 分钟）

> 各位同学，我们进入第 8 章——**软件本地化测试**。
>
> 前面几章我们一直在解决同一个问题的不同侧面：代码里的单元对不对、模块拼起来通不通、系统整体功能对不对、性能和安全扛不扛得住。这些工作有一个共同的前提，你可能从来没注意过——**我们默认这个软件是给中国人用的**：界面是中文，日期是年-月-日，货币是人民币，姓名是姓在前、名在后。
>
> 这一章把这个前提掀掉。同一个软件装到阿拉伯语用户的机器上，界面要从右往左排；装到德语用户的机器上，同一句话会长出一截，按钮装不下；装到美国用户的机器上，2/3/2009 到底是二月三日还是三月二日，居然说不清。这都不是"功能对不对"的问题，是"**换个地方还能不能用、用得顺不顺**"的问题。
>
> 这一章回答三件事：国际化 i18n 和本地化 l10n 差在哪、本地化测试要查什么、这些测试怎么组织和执行。
> 带着一个问题听：**你的软件如果明天要卖到德国和沙特，你手上的测试用例，有几条还能直接用？**

> **板书：**正中写"第 8 章 软件本地化测试"；左侧竖写"前面各章：默认说中文（对错・性能・安全）"；右侧

写"第 8 章：换个国家还行不行（i18n / l10n）"。

提问：一个在国内用得好好的软件，直接拿到国外卖，你猜第一批用户投诉会集中在哪些地方？

预设回答：

- 答"翻译错了、翻译得很怪"——这是最直接的一类，但它只是本地化问题的一部分，后面会看到还有格式、布局、文化三类。

- 答"界面乱码、日期看不懂、钱的面额不对"——很准，这三条正好对应字符集、数据格式两个技术板块。

- 答"把界面翻译成英文就能卖了"——这是最典型的错答。翻译只是 L10N 六项工作里的一项，代码没做国际化改造、日期和货币仍是硬编码，翻译完了照样用不了。

- 收口：所以本章的顺序是——**先有国际化能力，才谈得上本地化质量**。

易错点 / 考点：① 本章两条主线必须分清：i18n 是"让软件具备被本地化的能力"，l10n 是"把它变成某个地区的版本"；② 本地化不等于翻译，这是全章反复考的判断；③ 判断题陷阱："本地化测试只在测试阶段做"——错，它从设计、编码就开始埋下成败。

过渡：先不谈定义，我们看几个你我都遇到过的生活场景，你就知道这一章为什么必须单独成章。

【8.2】国际化生活的体验（3 分钟）

这一页课件很客气，只列了四个小场景，但这四个场景把本地化测试的全部难点都点到了。我们一个一个过。

第一个，**去美国旅行，最感不适应的是什么**。不是语言，是那些"看起来一样、规则不一样"的东西：插座电压、温度用华氏度、距离用英里、重量用磅、纸张尺寸不一样、星期一算一周的第一天还是星期日算第一天。这些都不是对错问题，是**约定不同**。软件的本地化，本质上就是把这些约定一条条对齐。

第二个，**2/3/2009 代表哪一天**。这个问题值得停三秒。美国人写月/日/年，所以是二月三日；欧洲人、很多亚洲国家写日/月/年，所以是三月二日。同一串数字，两个答案，**而且两个都"对"**。这就是本地化测试里最阴的一类缺陷：程序没报错、没崩溃、数据也存进去了，只是含义错了。用户看到账单日期差一个月，这在金融系统里是事故。

第三个，**George Bush 属于哪个家族**。这个问题问的是姓名和称谓。中文习惯姓在前，英文习惯名在前；中文叫"布什先生"，英文叫 Mr. Bush，称谓的位置反过来。还有音译问题：Bush 译成"布什"，同一个家族的人

译法要一致，否则检索、排序、去重全乱。看似小事，客户档案系统里一个姓名规则错了，几十万条数据就得重洗。

第四个，**收到邮件出现乱码**。这是字符编码问题，也是国内用户最熟悉的一种本地化故障。乱码的本质是"**写的时候用一种编码，读的时候用另一种编码**"。

四个场景归成一句话：**本地化测试要防的，不是程序算错，而是程序按别人的规矩算对了、按你的规矩算错了。** 这句话是本章的总纲。

板书：中间写"国际化生活的体验"；四角写四个场景：① 美国旅行（度量衡・纸张・一周起点）② 2/3/2009（日期顺序歧义）③ George Bush（姓名・称谓・音译）④ 邮件乱码（编码不一致）。底部一行大字："不是算错，是规矩不同"。

提问：2/3/2009 这个日期，如果让你在程序里存下来，你怎么保证不产生歧义？

预设回答：

- 答"按美国习惯存，月/日/年"——那欧洲用户填进来的日期就会被读错，方向反了。

- 答"用四位年-两位月-两位日的写法"——对，这是国际上最不容易误读的写法，把"年"放最前面，因为年的位数最多、最不可能被当成日或月。

- 答"加个下拉框让用户选格式"——可以缓解输入歧义，但存储层如果还是字符串，导出、排序、跨系统交换时又会出问题，治标不治本。

- 追问收口：真正可靠的做法是**存储用无歧义格式，显示用本地格式**，两个层面分开管。这个"存储与显示分离"的思想，后面在数据格式那节还会再讲一次。

易错点 / 考点：① 日期歧义是本地化案例题的最爱，答题要点是"**显示本地化、存储标准化**"；② 姓名问题的三个子项：顺序、称谓位置、音译一致性，考试常只答"顺序"而丢分；③ 乱码的根因一句话：**编码与解码用的字符集不一致**，不是"字库坏了"。

过渡：光说生活场景还不够，我们看两张真实的界面，同一个软件在两个地方长什么样。

【8.3】具体例子（2 分钟）

这一页是本章的第一组对照素材，请你带着"找不同"的心态看：**同一个软件、同一份数据，在中文简体的北**

京环境和英文的纽约环境里，呈现出来的东西差在哪。

先看差异的类型，不要急着看具体字。第一类是**文字本身**：界面上的词换了语言，这是最容易被发现的。第二类是**格式**：日期怎么写、时间用 12 小时制还是 24 小时制、钱用什么符号、千分位用什么符号、小数点用点还是逗号。第三类是**排列**：谁在前谁在后——日期是年月日还是月日年，姓名是姓在前还是名在前，地址是从大到小还是从小到大。第四类是**度量**：长度、重量、温度、纸张。

这四类里，只有第一类会被用户立刻发现并且投诉；**后三类才是最贵的**，因为它们不报错。举个真实感很强的场景：一份报表在北京导出是"1,234.56"，在纽约导出可能变成"1,234.56"看着一样，但到了德语环境就可能是"1.234,56"——逗号和点的含义整个反过来了。如果下游还有个程序去解析这个字符串，它会把一千二百三十四读成一点二三。

所以看这组对照图，我要求你练一个动作：**每看到一处差异，就问一句"这个差异是显示层的，还是会流进数据层的"**。只停在显示层的，改起来便宜；流进数据层、存进数据库、传给下游系统的，就是缺陷，而且往往是发布以后才炸的缺陷。

再补一句方法上的话：这类对照不要靠眼睛扫，要**列成清单逐项过**——日期、时间、数字、货币、度量、姓名、地址、电话、排序。这张清单就是后面 8.3.1 那节要展开的六项数据格式。

> **板书：**左右两栏："中文简体・北京"｜"英文・纽约"。中间画四条横线，分别标：文字｜格式｜排列｜度量。底部写一句："显示层差异 = 便宜；数据层差异 = 缺陷"。

提问：这两张图里的差异，哪些属于"改翻译文件就能解决"，哪些改翻译文件解决不了？

预设回答：

- 答"文字变了的那种，改翻译文件就行"——对，这是资源文件层面的工作。

- 答"日期和钱的写法，也能在翻译文件里改"——错。格式是**代码按区域设置算出来的**，翻译文件里改不了；代码里写死了格式串，就得改代码。

- 答"说不好，得看它是怎么实现的"——这个回答最专业。判断依据就是：这个值是**格式化出来的**，还是**拼字符串拼出来的**。拼出来的，本地化必出问题。

- 收口：所以本地化测试看到一个可疑显示，第一反应应该是"**它是怎么生成的**"，而不是"翻译得对不对"。

易错点 / 考点：① 四类差异（文字／格式／排列／度量）要能分类举例；② 常考判断："界面文字都翻译了，本地化就完成了"——错，格式类问题一条都没解决；③ 记住"显示层／数据层"这条分界，它是判断缺陷严重

度的实用尺子。

过渡：中文和英文好歹都是从左往右读，接下来这个例子会把方向也翻过来。

【8.4】示例—阿拉伯语（2 分钟）

这一页是阿拉伯语的界面示例。阿拉伯语属于**从右到左**的语言，也就是常说的 RTL（Right-to-Left）。这一页要你看出来的，不是"这些字我不认识"，而是**整个界面的空间逻辑被镜像了**。

具体镜像在哪些地方？第一，**阅读方向反过来**：文字从右边起头，段落右对齐。第二，**控件顺序反过来**：原来左边的导航跑到右边，原来右边的滚动条跑到左边，进度条的起点在右边。第三，**标点和成对符号要翻转**：左括号和右括号在视觉上要对调，问号、逗号的位置也跟着变。第四，**数字和混合文本更麻烦**：阿拉伯文里夹着英文单词、夹着数字的时候，会出现"双向文本"的排序问题，一段话里既有右到左又有左到右，边界在哪儿、数字该按哪边读，都需要算法去判断。

为什么这件事对测试特别重要？因为它**不可能靠翻译解决**。你把界面文字换成阿拉伯语，如果布局代码是按"左对齐、左边距、控件从左往右排"写死的，翻完就是一场灾难：文字压在图标上、按钮顺序颠倒、进度条反着走。这类问题在中文和英文环境里**永远暴露不出来**，只有真的切到 RTL 语言才会出现。

所以业界有个做法：在做国际化改造时就把布局写成"**跟着语言方向走**"——该用起始边、结束边的地方，不要写死左右。测试的时候也一样，别只测中文和英文，至少挑一个 RTL 语言过一遍，把镜像逻辑验掉。这一条，是很多团队第一次出海时最惨的教训。

板书：中间写"RTL：从右到左"；四支箭头分别写：阅读方向 → 右起・右对齐；控件顺序 → 导航/滚动条镜像；成对符号 → 括号翻转；混合文本 → 双向排序。底部："翻译解决不了方向，布局要跟着语言走"。

提问：把界面文字全部换成阿拉伯语，但布局代码一行不改，你预计最先崩的是哪个地方？

预设回答：

- 答"文字和图标会重叠"——很可能，因为文字变长、方向反了，原来的留白全对不上。

- 答"滚动条跑到左边，用户会找不到"——对，控件镜像没做就是这样。

- 答"应该没什么问题，字体能显示就行"——这是典型错答。方向、对齐、控件顺序全都要单独处理，光能显示字符离"能用"差得远。

- 追问收口：判断一个界面有没有真正支持 RTL，最省事的办法是看它**有没有把"左/右"换成"起始/结束"**这种与方向无关的表达。

> **易错点 / 考点：** ① RTL 的四个镜像点要能说出来；② 常考判断："支持双字节字符就等于支持阿拉伯语"——错，阿拉伯语是字符集问题加方向问题两件事；③ 记住术语 RTL／LTR，写简答题时用术语比用"从右往左"更得分。

> **过渡：** 场景看完了，我们回到教材的顺序，先把这一章的地图铺开。

【8.5】8.1 什么是软件本地化（1 分钟）

> 这一页是本章的四块内容，我按顺序点一遍，你记结构。
>
> **8.1 什么是软件本地化**——定义与基础。要分清国际化 i18n 和本地化 l10n 的关系，要讲字符集、国际化标准、本地化的基本步骤，最后落到"本地化测试包含哪几类工作"。这一块是地基，占的时间最长。
>
> **8.2 翻译验证**——翻译是本地化里最显眼的一项工作，但"翻完了"和"翻对了"是两件事，这一节讲怎么验证翻译，以及翻译之外的文化层面要改什么。
>
> **8.3 本地化测试的技术问题**——三块：数据格式、页面显示和布局、配置与兼容性。这是本章最"干货"的部分，几乎每条都能直接变成一条测试用例。
>
> **8.4 本地化的功能测试**——把本地化版本当成一个完整软件去测：集成、索引排序、联机文档，以及"本土测试"和"翻译验证测试"这两种典型的组织形式。
>
> 四块有一条共同的主线：**本地化测试查的不是"程序算得对不对"，而是"程序在别人的规矩下还对不对"**。所以我们既要懂测试，也要懂一点别人的规矩。

> **板书：** 竖排四块：8.1 什么是本地化（i18n/l10n・字符集・标准・步骤・测试范围）→ 8.2 翻译验证（翻译内容・文化心理・特殊符号）→ 8.3 技术问题（数据格式・显示布局・配置兼容）→ 8.4 功能测试（集成・排序・文档・本土测试・TVT）。右侧一句："别人的规矩"。

> **提问：** 这四块里，你觉得哪一块最难自动化、最依赖人？

> **预设回答：**

- 答"8.2 翻译验证"——有道理，语言好不好要人判断，机器只能查术语一致性。

- 答"8.3 技术问题"——也对，很多是"看着别扭"，判据难写。

- 答"都能自动化，写脚本扫一遍就行"——错。格式、编码、布局的**检测**可以自动化，但"这个译法在这个文化里合不合适"，脚本给不出答案。

- 收口：所以本地化测试的典型配置是**工具做筛查、人做判断**，这一点和性能测试很像。

> **易错点 / 考点：** ① 四块的名称与顺序；② 8.3 与 8.4 容易混：8.3 是"技术问题清单"，8.4 是"把本地化版本当完整软件测的组织方式"；③ 简答题常问"本地化测试包括哪些内容"，标准答案是 8.1.5 那六类，不要拿 8.3 的三块去答。

> **过渡：** 先看 8.1 这一节内部的路线，五小节，一层比一层落地。

【8.6】8.1 什么是软件本地化（1 分钟）

8.1 这一节分五小节，我给你画一条从"概念"到"能干活"的路线。

8.1.1 软件本地化与国际化——先把 i18n 和 l10n 这两个词掰开。这是全章最容易混的一对概念，也是考试必考的一对。

8.1.2 字符集问题——从 ASCII 讲到 Unicode、UTF-8。为什么要在这个位置讲编码？因为乱码是本地化故障里出现频率最高、用户最不能忍的一类，而它的根子全在字符集。

8.1.3 软件国际化标准——五条设计要求：语言切换机制、与语言无关的输入输出接口、资源文件国际化、包容本地数据格式。这五条是"一个软件够不够格被本地化"的检查表。

8.1.4 软件本地化基本步骤——八步流程，从配置管理、术语表，到提取、翻译、检入、编译、调 UI、再测试。这一节回答的是"本地化这件事怎么组织"。

8.1.5 软件本地化测试——落到六类测试工作：功能性、翻译、可用性、兼容性、文化宗教、手册验证。

一句话串起来：**概念分清楚（8.1.1）→ 底层看编码（8.1.2）→ 设计看标准（8.1.3）→ 执行看流程（8.1.4）→ 验收看六类测试（8.1.5）**。

> **板书：** 五格横排：8.1.1 概念（i18n/l10n）→ 8.1.2 编码（ASCII・Unicode・UTF-8）→ 8.1.3 标准（五条）→ 8.1.4 流程（八步）→ 8.1.5 测试（六类）。下方一行："概念 → 底层 → 设计 → 执行 → 验收"。

提问：为什么讲本地化测试，要花一整节去讲字符编码？

预设回答：

- 答"因为乱码最常见"——对，这是最直接的理由。

- 答"因为编码不对，后面的测试全是假的"——这个更深一层：编码错了，你测出来的"翻译不准确"可能只是显示错了。

- 答"编码属于开发的事，测试不用管"——错。测试人员至少要能判断"这是编码问题还是字体问题还是翻译问题"，否则提的缺陷单会被开发退回来。

- 收口：**能定位到层次，才提得出有效的缺陷**。

易错点 / 考点：① 五小节的顺序与关键词；② 8.1.2 与 8.3.2 的关系：前者讲原理，后者讲表现为乱码的现象与检查点；③ 别把 8.1.4 的"八步"和 8.4 的"本地化功能测试"混成一件事，前者是生产过程，后者是验证活动。

过渡：进入 8.1.1，先把 i18n 和 l10n 这两个词彻底分开。

【8.7】I18N vs. L10N（3 分钟）

这一页是本章最重要的一页，四个判断句，我一句一句抠。先解释这两个怪词怎么来的：i18n 是 internationalization，首字母 i 和尾字母 n 中间隔 18 个字母；l10n 是 localization，l 和 n 中间隔 10 个字母。业界图省事就这么缩写。再加一个词 globalization，缩成 g11n，它把 i18n 和 l10n 都包进去，指"全球化"这件整体的事。

第一句，**I18N 是 L10N 的基础和前提，为 L10N 做准备**。这句话定的是先后关系：没有国际化能力的软件，做本地化是硬改，成本极高。打个比方，i18n 是盖楼时把水电管道预埋好，l10n 是装修。管道没埋，装修的时候就只能砸墙。

第二句，**L10N 是 I18N 向特定本地语言环境的转换**。注意"特定"两个字。国际化是一次性的、通用的；本地化是多次的、一个地区一份。中文简体一份、中文繁体一份、德语一份、阿拉伯语一份，每一份都是一次独立的本地化。

第三句，**I18N 是软件产品源语言开发的一部分，属于 Engineering**。这句话很关键，它决定了责任归属：国际化是研发的活，写代码的时候就要做，不是测试阶段补的。你让测试去发现"这个字符串硬编码了"，测试

能发现，但改还是研发改。

第四句，**L10N 可以独立于 Engineering，可由第三方完成**。这就是为什么行业里有专门本地化公司：把资源文件发出去，翻译、回填、出多语言包。它像一个外包工序，可以独立发包、独立验收。

四句话合起来是一张组织图：**研发负责 i18n，本地化团队负责 l10n，测试两头都要验——验研发有没有留好接口，验本地化有没有填对内容。** 这也是本章后面所有内容的分工依据。

板书：三行对照：I18N＝internationalization（通用能力・Engineering・研发负责・一次做好）｜L10N＝localization（特定环境・可第三方・一地区一份）｜G11N＝globalization（i18n＋l10n 的总和）。中间一句箭头："i18n 是前提 → l10n 是转换"。底部："预埋管道 vs 装修"。

提问：一个软件先做了德语版，再想做日语版，需要重新做国际化吗？

预设回答：

- 答"需要，每种语言都要重新国际化"——错。国际化是通用能力，做一次就够，多做几遍是浪费。

- 答"不需要，只要再做一次本地化"——对。前提是当初的国际化做得干净：字符串抽出去了、格式没硬编码。如果当初是硬改出来的德语版，那日语版就得再硬改一遍，这时候"要不要补国际化"就成了成本决策。

- 答"看情况，得看第一版是怎么做的"——最专业的回答。这正是判断"技术债要不要还"的典型场景。

- 收口：所以第四句话的现实含义是——**i18n 做得好，l10n 是复制；i18n 做得差，l10n 是重做**。

易错点 / 考点：① 四句判断必背，尤其"i18n 属 Engineering、l10n 可第三方"这两句最容易考；② 别把 g11n 当成第三种技术，它是前两者的合集；③ 简答高频："简述国际化与本地化的区别与联系"——答题结构就是"前提与转换＋责任归属＋复用性"。

过渡：既然 i18n 是前提，那"具备国际化能力"具体指哪些能力？课件给了八条。

【8.8】I18N（3 分钟）

这一页是国际化的能力清单，八条。我按"数据、代码、界面、语言习惯"四组给你归一下，这样好记。

第一组，数据层：第一条，**支持 Unicode 字符集、支持双字节的字符**。这是地基中的地基。字符集不支持，后面所有翻译都是乱码。第二条其实也在数据层：**支持各个国家的度量衡、时区、货币单位格式等的设置**——注意它说的是"设置"，意思是这些不该写死，要能配。

第二组，代码层：分离程序代码和显示内容、消除 Hard code。这两条是一件事的两面：凡是要给用户看的文字，都不能写死在代码里，要放到资源文件里去。为什么？因为翻译的人不该去改源代码，改源代码就要重新编译、重新测，一轮下来成本翻几倍。第三条，**使用 Header files 去定义经常被调用的代码段**——课件这里说的是把常量的定义集中起来管理，好处是一处改动全局生效，不会漏改。

第三组，界面层：改善翻译文本尺寸，具有调整的灵活性。这条特别容易被忽略。翻译后的文字长度和原文不一样，界面必须留出伸缩余地——按钮能变宽、标签能换行、对话框能长大。如果界面是按英文字符数精确排版写死的，德语一上就崩。还有一条，**国际化用户界面设计（自我定义）**，意思是界面设计本身要考虑不同文化的阅读习惯和审美，不要默认全世界都跟你一样。

第四组，语言习惯层：支持各个国家的键盘设置——键盘布局不一样，快捷键的可用字符就不一样；**支持文字排序和大小写转换**——不同语言的字母表顺序不同，排序规则也不同，有些语言压根没有大小写。这两条在测试里非常容易漏，因为中文环境下你根本感知不到。

八条合起来是一句话：**国际化就是把"跟地区有关的一切"从代码里赶出去，变成可配置、可替换的东西。**

板书：四组纵列。数据层：Unicode/双字节｜度量衡・时区・货币可设。代码层：代码与显示分离｜消除硬编码｜头文件集中定义。界面层：文本尺寸可伸缩｜UI 设计国际化。语言习惯层：键盘设置｜排序与大小写转换。底部："把地区相关的一切赶出代码"。

提问：八条里，哪一条最容易被团队"以为自己做到了"?

预设回答：

- 答"消除硬编码"——很可能。界面上看得见的字符串容易抽出去，看不见的才要命：日志里的提示、异常信息、导出的文件名、报表表头。

- 答"文本尺寸可伸缩"——也非常典型。源语言下每个按钮都刚刚好，一翻译全挤在一起。

- 答"这些都做了，我们代码里没有中文"——这个回答本身就是陷阱。代码里没有中文，不等于没有硬编码的格式：`yyyy-MM-dd`、`$`、`kg`、`12/24 小时`，这些全是硬编码。

- 追问收口：所以检查硬编码，不能只搜中文，要搜**格式串和单位符号**。

易错点 / 考点：① 八条要能归组记忆（数据・代码・界面・语言习惯）；② 高频错点：把"消除硬编码"只理解成"界面文字"，漏掉日志、异常、文件名、格式串；③ 简答题若问"国际化包含哪些方面"，答这八条比答"翻译"得分高得多。

> **过渡：**清单是抽象的，我们看一张真正体现国际化能力的实例。

【8.9】国际化功能实例（2 分钟）

> 这一页是国际化能力的实例，也就是刚才那八条在真实界面上长什么样。你要练的是"**从现象倒推能力**"。
>
> 看第一类现象：**同一个数据，显示的样子随区域变**。比如一个数字，在一种区域设置下是 1,234.56，换一种设置可能变成 1.234,56——千分位和小数点的符号对调了。这说明程序没有自己拼字符串，而是调用了系统的区域格式。这就是"支持货币单位格式设置"落地的样子。
>
> 第二类现象：**日期时间的显示随区域变**，24 小时制和 12 小时制、星期几的写法、月份的名字。第三类现象：**排序结果随区域变**——同样一组人名，在两种语言下的先后顺序不一样，因为字母表顺序不同。第四类现象：**界面上不该出现的字符出现了**，比如本该显示一个字符的位置出现一个方框，这叫缺字形，属于字体或字符集问题。
>
> 怎么用这一页？我建议你把它当成一张"**验收照片**"：将来你测一个号称支持多语言的软件，就按这四类现象各挑一个点，切换区域设置，看它变不变。变了，说明能力在；不变，说明格式被写死了。
>
> 这里有个方法论要强调：**国际化测试最有力的手法就是"切换"**。切换区域、切换语言、切换时区、切换键盘，然后看同一个功能的表现是否随之改变。不切换，你测的只是源语言那一种情况。

> **板书：**四类现象竖排：数字格式随区域变｜日期时间随区域变｜排序随区域变｜缺字形出现方框。右侧写："手法：切换区域 / 语言 / 时区 / 键盘"。

> **提问：**你把系统区域从中文切到德语，发现日期显示没变，这算缺陷吗？

> **预设回答：**

- 答"算，说明格式写死了"——大概率算，但要先排除一种情况：这个字段可能是**业务要求固定格式**的，比如对外的接口报文。

- 答"不算，可能是缓存"——有这个可能，重启或清缓存再验一次，这是应有的严谨。

- 答"不算，日期格式无所谓"——错。日期格式在跨地区场景下会直接导致误读，前面 2/3/2009 那个例子已经说明后果。

- 收口：所以提这类缺陷时，缺陷单里要写清"**在什么区域设置下、期望什么格式、实际什么格式**"，三个要素缺

一不可。

易错点 / 考点：① "切换"是国际化测试的核心手法，要能说出切换哪四个维度；② 判断"显示不变"是不是缺陷，要先确认该字段有没有"固定格式"的业务要求；③ 缺字形（方框、豆腐块）与乱码是两种不同现象，考试常混。

过渡：i18n 的能力说完了，接下来看另一半——l10n 到底要做哪些事。

【8.10】L10N（3 分钟）

这一页是本地化的工作内容，六项。注意它不是"翻译"一项，而是六项，这个数字本身就是考点。

第一项，**翻译**。这是主体工作量，包括界面文字、提示信息、错误信息、帮助文档。翻译要讲术语一致、语气统一，后面 8.2 会专门讲怎么验证。

第二项，**地区文化、宗教**。这一项最容易被工程团队忽略，也最容易出事。图标、颜色、手势、动物形象、节日、地图边界，在不同文化里的含义可能完全相反。有些内容在某个国家是禁忌，你不改，产品就上不了架，甚至引发公关危机。

第三项，**度量衡和时区等**。公里还是英里、摄氏度还是华氏度、千克还是磅；时区不只是显示偏移，还包括夏令时——有些国家有、有些没有，切换的那一周特别容易出日期错乱。

第四项，**软件用户界面（UI）**。翻译完了要重新排版：控件变宽、对话框变高、菜单换行、图标里的文字要重画。这一项在 8.3.2 会展开。

第五项，**联机文档**，包括帮助文档和功能性的 PDF 文档。为什么强调"功能性 PDF"？因为有些软件的交付物本身就是 PDF——合同模板、报表模板、说明书，这些文件里的字体、编码、排版都要单独处理，而且往往不在研发的测试范围里，是最容易漏的一角。

第六项，**热键设置**。这一项很技术但很实际：菜单项的快捷键通常跟字母绑定，翻译之后字母变了，快捷键就得重新分配，还要避免和系统保留键冲突。中文用户习惯 Ctrl+C/V，德语用户键盘上 Z 和 Y 的位置都不一样，这些都要验。

六项记法：**翻译、文化、度量、界面、文档、热键**——前两个字能凑一句"翻文度界文热"，你按自己的方式记，关键是别只记翻译。

> **板书：**六行竖排：① 翻译 ② 地区文化・宗教 ③ 度量衡・时区 ④ UI ⑤ 联机文档（帮助・功能性 PDF）⑥ 热键设置。右侧批注："只做第①项＝没做本地化"。

> **提问：**六项里，哪一项最可能"研发完全不知道要做"？

> **预设回答：**

- 答"文化宗教"——很对，这属于产品与市场范畴，研发通常不碰。

- 答"联机文档里的 PDF"——也很对，PDF 往往由文档团队产出，测试根本不在链条上。

- 答"热键不用改，Ctrl+S 全世界都一样"——这是常见错答。功能键确实通用，但**菜单助记符**（按 Alt 后出现的那个带下划线的字母）是按语言分配的，翻译后必须重排，否则两个菜单项可能抢同一个字母。

- 收口：这六项的价值就在于提醒你——**本地化的验收面，比"界面翻译"宽得多**。

> **易错点 / 考点：**① 六项内容必背，"只做翻译"是典型错答；② 热键问题的准确说法是"菜单助记符需重新分配并避免冲突"，不是"功能键要改"；③ 常考判断："PDF 文档不属于软件本地化范围"——错，课件明确把它列进联机文档。

> **过渡：**为什么本地化里编码问题总是排第一？我们下到最底层，看字符集。

【8.11】8.1.2 字符集问题（3 分钟）

这一页讲字符集。课件给的定义很短：**字符集是操作系统中所使用的字符映射表**。"映射表"三个字是全页的关键——它说的是"**字符和数字之间的对应关系**"。计算机里没有"字"，只有数字；你看到的每一个汉字，本质上是一个编号。字符集就是这本编号簿。

课件列了五种：**ASCII**、**GB2312**、**ISO8859-X**、**Unicode**，以及 Unicode 的三种具体编码 **UTF-8 / UTF-16 / UTF-32**。它们的关系不是并列的，是**演进的**：ASCII 只装英文，一个字节，128 个位置；GB2312 是中国自己的国标，为了解决汉字，用两个字节；ISO8859-X 是欧洲的一堆单字节字符集，一个语系一份；Unicode 想一统天下，给全世界所有字符编号；UTF-8/16/32 是 Unicode 的三种"存法"。

为什么必须讲这个？因为**乱码的根因永远是同一句话：写的时候用 A 号簿，读的时候用 B 号簿**。举个最常见的场景：一个 UTF-8 保存的中文文件，被一个按 GBK 解码的程序打开，每个汉字都被拆成两半重新解释，出来的就是一串谁也不认识的字。反过来也一样。这不是"字库坏了"，是**两边用的编号簿不一样**。

更麻烦的是它会产生"**半个字符**"的问题。双字节字符占两个字节，如果程序在截断字符串时按字节切，正好切在一个汉字的中间，剩下的那半个字节谁也解释不了——这就是为什么本地化测试里，"字段长度限制"是个高危测试点：英文字段限 20 个字符，中文可能只能放 10 个字，按字节算又会切坏。

所以后面 8.3.2 讲乱码、8.3.3 讲 Non-Unicode 路径，根子都在这一页。这一页听懂了，后面那些现象都是它的表现。

板书：中间写"字符集＝字符↔数字的映射表"；下方一行演进：ASCII（1 字节・128 位）→ GB2312（2 字节・汉字）→ ISO8859-X（欧洲单字节）→ Unicode（全球编号）→ UTF-8/16/32（三种存法）。右侧红字："乱码＝写用 A 簿，读用 B 簿"。

提问：同一段中文，为什么在有的电脑上正常、在有的电脑上是乱码？

预设回答：

- 答"因为两台机器的系统语言不一样"——方向对，更准确的说法是**默认代码页或解码方式不一样**。

- 答"因为文件坏了"——错。文件一个字节都没变，变的是"怎么解释这些字节"。

- 答"装个字体就好了"——这是最典型的错答。字体只解决"有没有这个字形"，解决不了"这段字节代表哪个字"。乱码和缺字形是两件事。

- 追问收口：所以排乱码的顺序是——**先确认编码，再看字体，最后看翻译**。顺序反了，白折腾。

易错点 / 考点：① 字符集的定义与五种类型的演进关系；② 乱码与缺字形（方框、豆腐块）必须分清：前者是解码错，后者是字体缺；③ 高频考点："为什么双字节字符会导致字符串截断出错"——因为按字节切可能切在字符中间。

【思政落点 1】：字符集这一页最值得琢磨的地方是——**它出错的时候，程序一切正常**。没有异常、没有崩溃、日志也不报错，只是用户看到一堆乱码。这类问题考的是"**细节决定成败**"：一个编码参数、一个字节序、一次截断，看起来都是小事，可它直接决定了几亿用户能不能看懂你的界面。工程质量的差距，常常不在会不会写复杂算法，而在有没有人愿意为一个"看起来不影响运行"的字节较真。请大家养成这个习惯：**越是没人盯着的地方，越要自己盯。**

过渡：知道了有哪几种字符集，接下来看国际上用什么标准把它们管起来。

【8.12】字符集问题标准（3 分钟）

这一页列了五个标准，看着像背编号，其实每一个都在回答一个具体问题。我把"它管什么"讲清楚，编号你眼熟就行，考试很少要求你默写编号，但要求你知道**有哪几类标准、各管什么**。

第一个，**ISO/IEC 10646-1:2003**，它定义的是四字节编码的通用字符集，英文缩写 UCS，也叫"通用多八位编码字符集"。它回答的问题是：**全世界所有字符，怎么放进同一套编号体系里**。四个字节，空间足够大，这也是 Unicode 的编号基础。

第二个，**ISO 639-1:2002**，两字母语种代码标准。中文是 zh，英文是 en，日文是 ja，德语是 de。它管的是"**这是哪种语言**"。

第三个，**ISO 3166-1:1997**，国家代码标准。中国是 CN，美国是 US，台湾地区是 TW。它管的是"**这是哪个地区**"。

为什么语言和地区要分开？因为同一种语言在不同地区规则不同。中文在简体环境里用"软件"，在繁体环境里用"軟體"；日期习惯、货币、度量都可能不一样。所以本地化标识通常是"**语言 + 地区**"两段拼起来，这样才够精确。

第四个，**RFC 3066**，语言鉴定标签标准。它管的是"**这个标签怎么写才算合法**"，也就是把前面两个标准拼起来的语法规则。你在程序里看到的那些语言标识，依据的就是这一类标签规范。

第五个，**RFC 3629**，UTF-8。它管的是"**Unicode 怎么用变长字节存下来**"，下一页会专门讲。

五个标准记法：**编号（10646）、语言（639）、地区（3166）、标签（3066）、编码（3629）**。这五个词就是本地化标识体系的五根柱子。

再补一句考试口径：这五个标准里，测试人员真正会用到的只有两个——**知道语言代码和地区代码怎么拼出本地化目标，知道 UTF-8 有正式规范可查**。其余三个，你能说出它管什么就够了，不必背全名。

> **板书：**五行：ISO/IEC 10646-1:2003 → 通用字符集 UCS（4 字节编号）｜ISO 639-1:2002 → 语言代码（zh/en/ja/de）｜ISO 3166-1:1997 → 国家/地区代码（CN/US）｜RFC 3066 → 语言标签语法｜RFC 3629 → UTF-8。右侧一行："语言 + 地区 = 精确的本地化目标"。

提问：为什么本地化目标不能只写"中文"，要写成"中文简体"或"中文繁体"？

预设回答：

- 答"因为字不一样，一个简体一个繁体"——对，这是最直接的原因。

- 答"因为用词也不一样，比如软件和软体"——补得好，这就是术语层面的差异。

- 答"中文就是中文，一样的"——错。只写"中文"，翻译团队没法判断用哪套字形、哪套术语、哪套日期习惯，交付出来必然有一半用户不满意。

- 追问收口：同理，"英语"也要分英式美式——颜色是 colour 还是 color，日期顺序也不同。所以本地化目标的最小单位是"**语言 + 地区**"。

> **易错点 / 考点：** ① 五个标准各自"管什么"要能对上，别把 639（语言）和 3166（地区）搞反；② 课件给的是标准编号与版本年份，考试若考填空照抄，别自己改年份；③ 高频简答："为什么本地化标识要包含语言和地区两部分"。

> **过渡：** 标准讲完了，我们从最小的字符集看起——ASCII，所有编码故事的起点。

【8.13】示例：ASCII（2 分钟）

这一页是 ASCII 码表。这张表看着枯燥，但它是整个字符编码世界的起点，值得你花两分钟看懂它的结构。

ASCII 一共 128 个码位，用 7 个二进制位就能表示完，所以早期一个字节（8 位）的最高位是空着的——这个"空着的一位"后面会引出很多故事。表里的排列有规律，记住规律比背表有用：

前面一段是**控制字符**，编号最小的那批，它们不显示成字，而是控制行为——回车、换行、制表、响铃。你在程序里见的 \n、\t 就是它们。

中间是**数字**，从 0 到 9 是连续排列的，所以"字符 '7' 减字符 '0' 等于数字 7"这种写法才成立。

然后是**大写字母**，再是几个标点，然后是**小写字母**。大写和小写的编号差是固定的 32，这就是为什么大小写转换在英文里可以靠加减一个常数完成。

最后是英文标点符号。

看这张表要看出它的**局限**：128 个位置，装得下英文、数字、英文标点，**装不下任何一个汉字**。中文有几万个常用字，一个字节不够，必须两个字节。这就是 GB2312 出现的原因，也是"双字节字符"这个词的由来。

还有一条隐含信息：ASCII 里没有"€"、没有重音字母、没有阿拉伯文，欧洲人要写自己的语言就得扩展它——这就是 ISO8859-X 系列存在的原因。你把这条演进链记住：**一个字节不够 → 加字节 → 各国自己加 → 最**

后统一到 Unicode。

> **板书：**画一条横轴：控制字符 → 数字（连续）→ 大写字母 → 标点 → 小写字母 → 标点。上方标注"128 个码位 / 7 位"。下方红字："装不下汉字 → 双字节的由来"。右侧写演进链："1 字节 → 2 字节 → 各国扩展 → Unicode"。

提问：既然 ASCII 只用 7 位，一个字节还剩 1 位，为什么当初不直接用它装更多字符？

预设回答：

- 答"因为当时没考虑别的语言"——对，这是历史事实层面的回答，标准是按英语世界的需求定的。

- 答"因为剩下那位要用来做校验"——这是真实存在的一种做法，早期通信确实用最高位做奇偶校验，所以不能随便拿来装字。

- 答"因为 128 个够用了"——在当时够用，但正是这个"够用"给后来的国际化埋了坑。

- 收口：这段历史给我们的启示很实际——**标准的边界，往往就是后来所有兼容性问题的源头**。

易错点 / 考点：① ASCII 是 7 位、128 个码位，别写成 8 位 256 个；② 大小写差 32、数字字符连续，这两个规律常被用来出小题；③ 常考判断："ASCII 可以表示中文"——错，这是双字节讨论的起点。

过渡：ASCII 不够用，于是就有人想一次把全世界装进去——这就是 Unicode。

【8.14】示例：Unicode（3 分钟）

这一页讲 Unicode 的编号结构，课件给的信息不多，但每一条都是关键，我拆开讲。

第一，"**UCS-4，范围 0x0 到 0x10FFFF**"。这句话的意思是：Unicode 给每个字符一个编号，编号空间从 0 一直到十六进制的 10FFFF。为什么是这个范围？因为四个字节理论上能装到 FFFFFFFF，但没必要用那么多，划到这个上限足够覆盖现有和未来可预见的全部字符。

第二，"**1st 字节：0-16 个平面，后两个字节：0-FF 行，0-FF 列**"。这是一个坐标式的理解方式：整个编号空间被分成若干个"**平面**"，每个平面里有若干"**行**"，每行里有若干"**列**"。一个字符的位置就是"第几平面、第几行、第几列"。这种说法你听着像地图坐标，其实本质就是——**编号是分层的**。

第三，"**0 平面：BMP（Basic Multilingual Plane）＝ UCS-2**"。这是全页最重要的一句。第 0 个平面叫"基本多语言平面"，几乎所有日常用的字符都在这里：拉丁字母、汉字、日文假名、阿拉伯文、标点符号。所谓

UCS-2，就是"**用固定的两个字节表示这个平面里的字符**"。前面说的"双字节字符"，指的就是这一层。

由此可以推出一个很实用的结论：**绝大多数情况下，一个汉字占两个字节**（在 UCS-2 的世界里）；但在 UTF-8 里它占三个字节。同一句话，不同编码下字节数完全不同——这正是"按字节限制长度"会出事的原因。

再补一句：基本平面之外的字符（比如一些生僻字、古文字、表情符号）要用"代理对"来表示，也就是两个双字节拼起来。这类字符是本地化测试的高危区：长度算不对、截断截坏、排序乱掉，全从这儿来。

板书：画一个坐标示意：平面（0–16）× 行（00–FF）× 列（00–FF），范围标 0x0–0x10FFFF；在"平面 0"处写"BMP ＝ UCS-2（双字节区）"。下方一行："汉字：UCS-2 两字节 / UTF-8 三字节"。

提问：一个字段限长 10 个字节，能存几个汉字?

预设回答：

- 答"10 个汉字"——错，把"字节"当成"字符"了，这是最普遍的错误。

- 答"5 个汉字"——如果按 UCS-2 每字两字节算，对；但如果程序内部是 UTF-8，汉字占三个字节，就只能放 3 个。

- 答"要看编码方式"——最准确的回答。**先问编码，再算字数**，这才是专业反应。

- 追问收口：所以本地化测试遇到"长度限制"，第一件事是问清"**限的是字节还是字符**"，第二件事是拿最长的目标语言去试。

易错点 / 考点：① BMP、UCS-2、"双字节"三者的关系要能串起来讲；② 高频判断："一个汉字就是一个字节"——错，这是本地化缺陷的高发根源；③ 编号范围 0x0–0x10FFFF 常作为填空点，注意是 16 个平面。

过渡：Unicode 解决了"编号"，还没解决"怎么存"，这就是 UTF-8 要回答的问题。

【8.15】示例：UTF-8（2 分钟）

这一页讲 UTF-8，课件只给了一个 RFC 的链接，但内容必须讲清楚，因为它是今天互联网上最常用的编码。

UTF-8 的核心思想是"**变长**"：一个字符用几个字节，取决于它是哪个字符。英文的 26 个字母还是 1 个字节，和 ASCII 完全一样；带重音的欧洲字符 2 个字节；常用汉字 3 个字节；更生僻的、或者表情符号 4 个字节。

变长带来三个后果，每个后果都是一类测试点。

第一个后果，**向后兼容**：纯英文的文本，用 UTF-8 存和用 ASCII 存，字节完全一样。这就是为什么 UTF-8 能迅速取代各种本地编码——老系统不用改就能读。

第二个后果，**长度不再等于字符数**。这是最要命的一条。程序里所有"取前 N 个字符""截断到 N 字节""按长度校验"的地方，在 UTF-8 下都必须小心。截断如果正好切在一个三字节汉字的中间，剩下两个字节就是非法序列，显示出来就是乱码或者问号。

第三个后果，**解码可以自同步**：UTF-8 的设计让每个字节的高位能表明"我是单字节字符，还是多字节字符的第几个字节"。好处是即使数据流中间丢了一段，程序还能从下一个合法起点重新对齐，不会一路错到底。这个特性是它比很多老编码健壮的原因。

课件给的那个 RFC 链接，是 UTF-8 的规范原文。课上不要求你读，但你要知道一件事：**编码是有正式规范的，不是各家自己商量**。遇到编码争议，最后都要回到规范上说话。这也是一种职业习惯——**有标准的地方，就别靠猜**。

> **板书：**画一张对照表：英文 1 字节｜欧洲带重音字符 2 字节｜常用汉字 3 字节｜生僻字/表情 4 字节。右侧三条：向后兼容 ASCII｜长度≠字符数｜解码自同步。底部："截断切在字符中间 = 乱码"。

提问：一个用 UTF-8 存的文件，被一个只认 ASCII 的老程序读，会发生什么？

预设回答：

- 答"中文会变乱码"——对，非 ASCII 部分全部读不出来。

- 答"英文部分正常，中文变乱码"——更准确。UTF-8 的 ASCII 段兼容，所以英文毫发无损，只有多字节部分出问题。

- 答"文件会打不开"——不一定，多数情况是能打开但内容错，这种"看起来打开了其实是错的"才是最危险的。

- 收口：这正是为什么本地化测试里要专门有一条"**在非 Unicode 环境下的表现**"，8.3.3 会讲。

易错点 / 考点：① UTF-8 是变长编码，1–4 字节；② "长度≠字符数"是本地化缺陷的高发点，务必能举例；③ 常考判断："UTF-8 是 Unicode 的另一种字符集"——不准确，Unicode 是字符集与编号，UTF-8 是它的**编码方式**之一。

过渡：底层讲完，我们把视角抬到设计层——一个软件要做到哪五条，才算"具备国际化标准"。

【8.16】8.1.3 软件国际化标准（3 分钟）

这一页是国际化的五条设计要求。前面 8.1 那八条讲的是"能力清单"，这五条讲的是"**设计上的硬要求**"，两者角度不同：八条回答"要具备什么"，五条回答"接口做成什么样才算合格"。这五条我一条一条过，每一条我都会告诉你它对应的测试点。

第一条，**切换语言的机制**。软件要能换语言，而且换完不能要求重装、重启。这一条对应的测试点是：切换入口在哪儿、切完界面有没有立刻变、有没有一部分文字没跟着变（通常是缓存或者写死的地方）。

第二条，**与语言无关的输出接口**。意思是：程序往外吐的东西——日志、导出文件、报表、接口报文——格式不能依赖当前语言。为什么？因为一旦依赖，同一个系统在不同语言环境下产出的数据就不一致，下游系统解析不了。测试点是：同一份数据在两种语言环境下导出，结果该一样的必须一样。

第三条，**与语言无关的输入接口和标准的输入协议**。这条跟第二条对称。输入不能只认一种编码、一种格式。用户可能在中文输入法下敲全角符号，可能从别处粘贴一段 UTF-8 文本，程序都要能正确接收。测试点：全角半角混输、粘贴多语言文本、导入外来文件。

第四条，**资源文件的国际化**。所有要翻译的内容要放在资源文件里，而且要有一套管理机制——分文件、分模块、能增量更新、缺项能回退到源语言。测试点：某个资源项缺失时，界面是显示空白、显示键名、还是回退显示源语言？三种表现里，"显示键名"是最容易漏测的。

第五条，**支持和包容本地化数据格式**。这是数据层的总要求：数字、货币、日期、时间、度量衡，都要能按目标地区的习惯来，而不是硬编码成一种。

这五条合起来，就是"**界面可切、输出中立、输入宽容、资源可换、格式可配**"。你在评审一份国际化设计方案时，就拿这五句话去对。

> **板书：**五行：① 切换语言机制（不重装不重启）② 输出接口与语言无关 ③ 输入接口与语言无关・标准协议 ④ 资源文件国际化（缺失可回退）⑤ 包容本地格式。右侧竖排记忆句："界面可切・输出中立・输入宽容・资源可换・格式可配"。

提问：第四条"资源文件国际化"，如果某个词条忘了翻译，你希望界面显示什么？

预设回答：

- 答"显示英文原文"——这是比较常见的降级策略，用户好歹能猜。

- 答"显示空白"——最差的选择，用户看到一块空白，既不知道是什么也不知道出错了。

- 答"显示一个像 %s 或者 BTN_OK 的东西"——这是错答里最不理想的，用户完全看不懂；但对测试来说它反而**是好事**，因为它一眼就能被看出来，比"静默回退成英文"更容易发现漏译。

- 追问收口：所以验收时要定一个明确策略：**漏译时回退到源语言，并且让测试能定位到是哪一条漏了**。策略不定，测试就没法判对错。

> **易错点 / 考点：** ① 五条要求要能默写关键词；② 输入接口与输出接口是成对的两条，答题时别只写一半；③ 高频判断："国际化只是界面的事"——错，第二、三条明确要求输入输出接口与语言无关。

> **过渡：** 设计标准有了，接下来看测试怎么切分——i18n 测试和本地化测试，查的不是同一批东西。

【8.17】I18N 测试（3 分钟）

这一页是本章结构最清楚的一页，它把"国际化支持"和"本地化测试"两列并排放，一共十四个问号。这一页你要带走的是**分工**：左列查能力，右列查填充。

先看左列，**国际化支持**，七个问号：多字节字符集和脚本支持吗？多字节字符的输入和显示正常吗？多字节的文件名、文件夹名及其处理对吗？区域设置生效吗？索引和排序正确吗？本地化操作系统上能跑吗？键盘支持吗？

这七个问题的共同特征是：**它们都在源语言版本上就能测**。不需要等翻译交付。比如"文件名支持多字节"，你拿一个中文文件名在英文系统上建一下就知道；"区域设置生效"，你切换区域看日期变不变就知道。这就是 i18n 测试的最大优势——**早，而且便宜**。

再看右列，**本地化测试**，七个问号：本地化资源齐全吗？区域设置有没有被硬编码？链接和串联的字符串有没有问题？资源文件里有没有夹着非本地化内容？文字有没有扩展余地？非层次图像（就是图片里画死的文字）里的文字处理了吗？其它组件呢？

右列的共同特征是：**必须等翻译交付之后才能测**，而且它的缺陷往往更"贵"——因为改一处要重新翻译、重新排版、重新回归。

这一页还特别标了一条：**Pseudo-translation（伪翻译），重要部分**。什么叫伪翻译？就是在翻译还没交付之前，先把资源文件里的英文**机械地替换成一段变长的、带特殊字符的假文本**，然后跑一遍界面。它能在几分钟内暴露三类问题：文字扩展装不下、硬编码没抽出来、编码处理有问题。成本几乎为零，收益极高。

最后课件给了个术语：**MBC = Multiple Byte Character，多字节字符集**。以后看到 MBC 你就想到"中文、日文、韩文这类占多个字节的字符"。

板书：左右两列。左"国际化支持（源语言版就能测）"：MBC 字符集/脚本｜MBC 输入显示｜MBC 文件（夹）｜区域设置｜索引排序｜本地化 OS｜键盘。右"本地化测试（翻译后测）"：本地化资源｜区域设置硬编码｜链接/串联字符串｜资源含非本地化内容｜文字扩展余地｜非层次图像文字｜其它组件。中间标红："Pseudo-translation ＝ 提前引爆"，底部注 "MBC = Multiple Byte Character"。

提问：为什么"链接/串联字符串"会成为一个单独的测试项？

预设回答：

- 答"因为句子拼起来语序会乱"——对。程序里常写"共 " + 数量 + " 条记录"，翻译成别的语言后语序可能整个反过来，拼出来就是病句。

- 答"因为翻译只翻了片段，前后接不上"——也对，这是另一种表现：句子被切成几段分别翻译，译者看不到上下文，译文风格和术语就不统一。

- 答"字符串拼接没什么问题，能显示就行"——错。这是本地化最典型的隐性缺陷，源语言下读着通顺，换语言就成了机器翻译腔甚至语义错误。

- 收口：所以正确做法是**用带占位符的完整句子**，让译者看到整句；测试时要专门去找界面上那些"由几段拼出来的话"。

易错点 / 考点：① 两列各七个问号要能说出大意，重点是分清"哪一列能在源语言版上测"；② 伪翻译（pseudo-translation）是本章的高频名词，要能说出它的作用和成本优势；③ MBC 的英文全称常考填空。

过渡：既然左边那一列能提前测，具体用什么方法测？下一页给了三种。

【8.18】国际化测试方法（2 分钟）

这一页给的是国际化测试的三种方法，很简短，但每一种都对应一个明确的场景，我补上"什么时候用、怎么用"。

第一种，**设计评审和代码审查**。这是静态方法，不运行程序。评什么？评设计里有没有把格式、语言、地区这些东西写死；审代码里有没有硬编码字符串、有没有按字节截断、有没有用系统默认编码去读写文件。为

什么把它放第一位？因为这类问题**越早发现越便宜**。等到测试阶段发现"整个模块的日期格式都是拼字符串拼出来的"，改造成本可能是重写。

第二种，**针对源语言的功能测试，比如不同的区域设置、不同的时区显示**。注意这句话的用词——"针对源语言"。意思是：界面还是英文（或中文），但我把系统区域切到德国、把时区切到纽约，看功能还对不对。这是最划算的一类测试，因为它不需要任何翻译交付，却能暴露大量区域相关缺陷。时区那一项尤其值得测：跨时区、跨夏令时切换点、跨日期边界，最容易出"差一天"的 bug。

第三种，**针对伪翻译版本的测试**。伪翻译版本就是把资源文件里的文本换成变长的假文本，跑起来看界面。它专治三类病：布局被撑破、文字被截断、以及那些"没被抽进资源文件、所以没被伪翻译覆盖到"的硬编码——后者尤其宝贵，因为**凡是没变成假文本的地方，就是硬编码的地方**。这一招我叫它"反向定位"。

三种方法的关系是：**评审管"设计上对不对"，源语言测试管"区域相关功能对不对"，伪翻译测试管"界面和资源处理对不对"**。三层都做完，国际化这块基本就干净了。

> **板书：**三行：① 设计评审 + 代码审查（静态・最便宜）② 源语言功能测试（切区域/时区，不需翻译）③ 伪翻译版本测试（撑破・截断・反向定位硬编码）。右侧写："凡是没变成假文本的地方＝硬编码"。

提问：伪翻译测试为什么能"反向定位"硬编码？

预设回答：

- 答"因为假文本很显眼，一眼就能看出哪些没被替换"——对，这就是核心逻辑。界面上还留着正常英文的地方，就是没走资源文件的。

- 答"因为它会把所有文字变长，撑破的地方就是布局问题"——这是它的第二个作用，但不是"反向定位"。

- 答"伪翻译就是随便乱码一下，测不出什么"——错。伪翻译是**有规则地**替换：变长、加特殊字符、加边界标记，目的性很强，不是乱码。

- 收口：所以伪翻译既是"压力测试"（用变长文本压布局），也是"覆盖率探针"（找出漏抽的字符串）。

易错点 / 考点：① 三种方法要能按"静态/动态"分类；② 伪翻译的两大作用（压布局、找硬编码）是简答得分点；③ 常考判断："国际化测试必须等翻译完成才能做"——错，前两种方法都不需要翻译。

过渡：方法有了，具体查哪些点？下一页给了一张十项的清单。

【8.19】国际化测试点（3 分钟）

这一页是国际化测试的检查清单，十项，最后还有个省略号——意思是还能往下加。我按"输入、显示、转换、交互"四组给你归，每一项都说一句"怎么测"。

输入组：**多字节和单字节文字的混合**——在一句话里同时敲中文、英文、数字、全角符号，看显示和存储对不对；**输入法编辑器（IME）**——中文输入法的候选框、联想、上屏，在某些界面里会出问题，比如输入框被候选框挡住、上屏后光标位置跳错。

显示组：**双向识别功能**——就是前面讲的 RTL，一段文字里混了从左到右和从右到左两种方向，程序要能正确判断每一段的走向；**大小写转换、换行**——有些语言没有大小写，转换函数遇到它们会不会出错；换行规则也不一样，中文可以在任意字之间断行，英文必须按词断，日文还有自己的一套禁则。

转换组：**硬编码**——找那些没抽进资源文件的文字和格式；**语言切换方式**——切换入口、切换后是否立即生效、是否有残留没变的文字。

交互组：**快捷组合键**——翻译后菜单助记符要重新分配，避免冲突；**纸张大小**——A4 和 Letter 尺寸不同，打印排版会错位，这一项在打印报表、发票、合同类软件里是必测；**电话号码**——位数、分组、区号、国际前缀，各国都不同，输入校验如果按中国手机号写死 11 位，出国就废。

最后一项 **词序问题**——主谓宾的顺序、形容词和名词的先后、数量和单位的组合方式，各国都不一样。程序如果靠拼接字符串来组句子，词序一定出问题。

这十项我给你一个记忆抓手：**"混、法、切、硬、换、键、纸、电、序"**——混合输入、输入法、切换、硬编码、大小写换行、快捷键、纸张、电话、词序，再加一个双向识别。你把这十项抄在测试计划的第一页，做国际化测试时逐项打勾。

> **板书：**四组纵列。输入：多字节单字节混排｜IME。显示：双向识别｜大小写・换行。转换：硬编码｜语言切换方式。交互：快捷组合键｜纸张大小｜电话号码｜词序。底部："十项逐项打勾，别凭印象"。

提问：电话号码为什么会被列进"国际化测试点"？它不就是一串数字吗？

预设回答：

- 答"因为各国位数不一样"——对，中国手机 11 位，美国 10 位，有些国家还带前导零。

- 答"因为格式不一样，有的加括号有的加横线"——也对，显示格式和校验规则都要跟着地区走。

- 答"电话号码是业务字段，跟本地化没关系"——错。凡是有"固定长度、固定格式、固定校验规则"的字段，都是本地化的高危点：邮政编码、身份证号、税号、车牌，全是同一类问题。

- 追问收口：所以判断一个字段要不要本地化，就问一句"**这个规则在别的国家还成立吗**"。

> **易错点 / 考点：** ① 十项要能归类记忆，考试常以多选题形式出现；② "纸张大小"最容易被忘，记住它属于打印排版类；③ 高频案例："输入框写死手机号 11 位"是本地化缺陷的经典题。

> **过渡：** 清单讲完了，我们看一组截图，把这些抽象的点落到眼睛里。

【8.20】示例（2 分钟）

这一页是把上一页那十项落到具体界面上的示例。看图的方法我在第 3 页给过：**每看到一处异常，先判断它是哪一类问题，再判断它是显示层还是数据层**。这一页我带着你走一遍这个判断流程。

第一类，看**文字装不装得下**。界面上如果有文字被截断、有省略号、有半个字露在外面，这就是布局问题，属于 8.3.2 的内容。判断要点：是控件太小，还是文本太长？如果是控件按源语言长度写死的，那是设计问题，不是翻译问题。

第二类，看**字符显不显示得出来**。有方框、有问号、有一串怪字符，这是编码或字体问题。这里要分清：方框（豆腐块）通常是字体缺字形；成串的怪字符通常是解码用错了字符集。两者修法完全不同。

第三类，看**方向和对齐**。RTL 语言该右对齐的地方左对齐了、镜像没做、括号反了，都属于这一类。

第四类，看**语言混排**。中英文数字混在一起时，间距、基线、换行位置是否正常。这一条最考验实现质量，也最难自动检测。

我想强调一个职业习惯：**看截图不能只看"哪里不对"，要能说出"为什么不对、应该改成什么"**。缺陷单里写"界面显示异常"是没法处理的；写"德语环境下'保存'按钮文字超出边框约两个字符宽度，建议按钮宽度自适应或允许换行"，开发一看就知道怎么改。**缺陷描述的精度，就是测试人员的专业度。**

> **板书：** 四类判断：装不装得下（布局）｜显不显示得出（编码/字体）｜方向对齐（RTL）｜语言混排（间距基线换行）。右侧写缺陷描述三要素："现象 + 环境 + 建议"。

> **提问：** 截图里一个按钮上的字被切掉了，你怎么判断是翻译太长还是控件太小？

> **预设回答：**

- 答"把语言切回源语言看是否正常，正常就是翻译太长"——这是最快的判断法，很实用。

- 答"看控件有没有固定宽度"——这是从实现角度的判断，更根本。

- 答"反正就是显示问题，让开发自己看"——错。不区分原因，开发可能只改翻译（缩短译文），而真正的问题是控件写死了宽度，换个更长的语言还会犯。

- 收口：所以好的缺陷单会写"**建议控件宽度自适应，而不是缩短译文**"——这就是测试人员给出的专业判断。

易错点 / 考点： ① 四类问题的分类判断是案例题的基本功；② 缺陷描述"现象+环境+建议"三要素，考试若给一段糟糕的缺陷描述让你改，就按这三要素补；③ 别把"字体缺字形"和"编码错误"混为一谈。

过渡： 从这一页开始进入流程——本地化这件事，从头到尾分八步。

【8.21】8.1.4 软件本地化基本步骤（3 分钟）

这一页是本地化的八步流程。这是本章唯一的完整流程，考试很可能让你排序或者补步骤，所以我按"准备、提取、翻译、回填、验证"五个阶段给你串一遍。

准备阶段，两步。第一步，**建立配置管理体系，跟踪目标语言各个版本的源代码**。为什么这一步排第一？因为本地化是"多语言、多版本并行"的活：中文版在改、英文版在改、德语版也在改，如果没有配置管理，你根本说不清"这个德语版是从哪个源版本翻出来的"。一旦源语言改了而译文没跟上，就会出现"新功能没翻译"的缺口。第二步，**创造和维护术语表**。术语表是本地化的命根子：同一个词在全产品里必须统一译法，否则用户在界面、帮助、文档里看到三种叫法，会以为是三个不同功能。

提取阶段：**源语言代码和资源文件分离、或提取需要本地化的文本**。这一步做得好不好，直接决定后面所有工作的成本。提取不干净，译者就会拿到一堆不该翻的东西（变量名、日志、内部代码），翻完了还会被误用。

翻译阶段：**把分离或提取的文本、图片等翻译成目标语言**。注意课件写的是"文本、图片等"——图片里的文字也要处理，这就是前面说的"非层次图像的文字"，往往要重画图。

回填阶段：**把翻译好的文本、图片重新检入目标语言的源代码版本**。这里的关键词是"检入"——走配置管理，不是手工粘贴。

验证阶段，三步：**如果需要，编译目标语言的源代码；测试翻译后的软件，调整 UI 以适应翻译后的文本**——

注意这一步是"调 UI"，属于返工；**测试本地化后的软件，确保格式和内容都正确**——这一步才是完整的功能验证。

八步走完你会发现：真正"翻译"只占一步，其余七步都是**工程和管理**。这就是为什么本地化项目的成败，往往不取决于译者水平，而取决于流程。

板书：五阶段八步：准备（配置管理・术语表）→ 提取（代码与资源分离）→ 翻译（文本・图片）→ 回填（检入源版本）→ 验证（编译・调 UI・测格式与内容）。右侧批注："翻译只占 1/8，其余是工程与管理"。

提问：为什么"术语表"要放在翻译之前，而且要求"维护"而不只是"创造"？

预设回答：

- 答"因为术语不统一，用户会以为功能不一样"——对，这是最直接的用户体验后果。

- 答"因为翻译公司可能换人，得有依据"——很实际。人员流动、外包更换，术语表是唯一能保证一致性的东西。

- 答"术语表是翻译的事，测试不用管"——错。测试恰恰要用术语表当判据：译文里出现的词，和术语表不一致，就是缺陷。

- 追问收口：所以测试团队要拿到术语表，把它变成一条可执行的检查规则。**没有术语表，翻译质量的判定就只能靠"感觉"**。

易错点 / 考点：① 八步的顺序与名称，尤其"调 UI"在"完整验证"之前，别颠倒；② 高频判断："本地化的主要工作是翻译"——错，八步里只有一步是翻译；③ 术语表的作用（一致性 + 可判定）是简答常见得分点。

【思政落点 2】：这八步里最不起眼的是第一步——配置管理。它不产出任何看得见的东西，却决定了后面七步有没有依据。现实中的本地化事故，很多不是译错了，而是"**译文对的，但对应的是上一个版本的界面**"。这类问题靠聪明解决不了，只能靠流程：谁改的、什么时候改的、改在哪个分支、对应哪一版译文，每一步都留痕。请记住：**严谨的流程意识，是让普通人也能稳定交付高质量的唯一办法**。做测试的人，尤其要尊重流程——因为流程保护的正是"不出事"这件事本身。

过渡：八步是"做事"的顺序，下一页把这些步骤放进整个产品生命周期里看。

【8.22】本地化过程（2 分钟）

这一页把本地化放进产品的大流程里，五个环节加一条主线：**国际化软件设计 → 核心功能测试 → 国际化测**

试 → 翻译测试 → 本地化测试 → 全球化软件发布。

注意顺序里的两个关键点。

第一，**国际化软件设计排在最前面**。这印证了前面那句话：i18n 属 Engineering，是设计阶段的事。设计没做，后面的测试只能发现缺陷，不能创造能力。

第二，**核心功能测试在国际化测试之前**。为什么？因为要先确认"这个软件在源语言下功能是好的"。如果功能本身就有问题，你去测国际化，测出来的失败你分不清是功能缺陷还是本地化缺陷。这就是测试里最基本的一条纪律：**先确认基线，再测增量**。

然后是**国际化测试**——查能力，能不能被本地化。接着是**翻译测试**——检查译文本身：完整性、术语、语境。最后是**本地化测试**——把翻译好的版本当完整软件测一遍，包括安装、升级、打印、帮助文档。

这条链子最后落在**全球化软件发布**上。注意"发布"也是一个环节而不是终点：多语言版本要一起发、版本号要对齐、语言包要能独立更新。

我给你一句总结：**这条链子上，越靠前的环节改起来越便宜，越靠后的越贵。** 设计阶段改一行配置，成本是一；发布之后发现方向错了，成本是重做。这就是为什么本章要花那么多篇幅讲 i18n 的设计标准。

还有一个现实要提醒你：这条链子上的每个环节都会「回流」。翻译测试发现译文和界面对不上，要退回到回填；本地化测试发现某个控件装不下，要退回到调 UI 再重新构建。所以真实的本地化项目不是一条直线，而是一条带着回环的螺旋。你将来写计划，一定要给返工留出时间，别把排期排成直线。

> **板书：**一条横向流程：国际化软件设计 → 核心功能测试 → 国际化测试 → 翻译测试 → 本地化测试 → 全球化软件发布。流程下方画一个从低到高的成本箭头："越靠后越贵"。

提问：为什么"翻译测试"要单独成一个环节，而不是合到"本地化测试"里？

预设回答：

- 答"因为翻译问题太多，需要单独集中查"——有道理，译文问题量大且集中在语言层面。

- 答"因为做的人不一样，翻译测试常由语言专家做"——很准确，这是组织上的原因，也呼应了"l10n 可第三方完成"。

- 答"没必要分开，一起测就行"——在很小的项目里可以合并，但一旦多语言并行，混在一起会导致缺陷归属不清：到底是译文错，还是程序把译文显示错了？

- 收口：分开的目的，是让**缺陷归因**变得干脆：语言问题给译者，显示问题给开发。

> **易错点 / 考点：**① 六个环节的顺序必背；② "核心功能测试在本地化之前"这条纪律常考；③ 高频判断："本地化测试可以替代翻译测试"——错，两者的判据和责任人不同。

> **过渡：**链条走完了，我们回头看 8.1 的最后一小节：本地化测试到底包含哪六类工作。

【8.23】8.1.5 软件本地化测试（3 分钟）

> 这一页是本地化测试的六类工作，也是本章最可能出简答题的地方。我一条一条讲，每条都给你一个"具体查什么"的例子。
>
> 第一，**功能性测试**，包括所有基本功能、安装、升级等测试。这一条最容易被跳过——很多人以为"功能在原版测过了，本地化版不用重测"。错。本地化版是**重新编译过的另一个产品**：资源文件换了、有时候代码也动了，安装包不一样、升级路径不一样。安装和升级尤其要测：语言包能不能装、装错了语言能不能切、升级之后译文会不会被覆盖回英文。
>
> 第二，**翻译测试**，包括语言完整性、术语准确性等的检查。完整性指有没有漏译、有没有多余的英文残留；准确性指术语是否统一、语气是否合适、有没有机翻痕迹。
>
> 第三，**可用性测试**，包括用户界面、度量衡和时区等。注意课件把"度量衡和时区"放在可用性里——这不是笔误。对用户来说，看到"3 磅"还是"1.4 公斤"，是好不好用的问题；看到时间差 8 小时，也是好不好用的问题。
>
> 第四，**兼容性调试**，包括硬件兼容性、版本兼容性等。这里的关键词是"调试"——本地化环境下的兼容问题往往需要反复试：不同版本的操作系统、不同的语言包、不同的输入法、不同的打印机驱动。
>
> 第五，**文化、宗教、喜好等适用性测试**。这一条最软，但杀伤力最大：地图边界、宗教符号、颜色寓意、手势、节日、历史表述。这类问题一旦出错，不是 bug 而是事故，可能直接导致产品下架。
>
> 第六，**手册验证**，包括联机文件、在线帮助、PDF 文件等测试。文档里的截图是不是旧版本、链接能不能跳、索引能不能搜、PDF 里的字体有没有嵌进去，都要验。
>
> 六类记法：**功能、翻译、可用、兼容、文化、手册**。前四类是技术活，后两类是判断题——尤其第五类，考的是你的见识和分寸。

> **板书：**六行：① 功能性（基本功能・安装・升级）② 翻译（完整性・术语准确性）③ 可用性（UI・度量衡・时区）④ 兼容性（硬件・版本）⑤ 文化宗教喜好 ⑥ 手册（联机文件・在线帮助・PDF）。右侧写："本地化版是重新编译的另一个产品"。

> **提问：**为什么"安装和升级"必须重测？本地化版和原版在安装上有什么不同？

> **预设回答：**

- 答"因为安装包不一样，里面多了语言文件"——对，这是最直接的原因。

- 答"因为要测语言选择、语言切换"——也对，安装向导本身也要翻译，而且要在安装时就能选语言。

- 答"安装程序没改，不用测"——错。安装向导里的文字也要翻译；安装路径里可能有非 ASCII 字符；升级时旧版本的配置会不会覆盖新译文，这些全是新风险。

- 追问收口：所以本地化测试的第一条用例，往往是"**在一台目标语言的干净机器上，从零装一遍，再升级一遍**"。

> **易错点 / 考点：**① 六类工作必背，顺序不必死记但内容不能漏；② "度量衡和时区"归在可用性里，这是课件的归类，答题时按课件口径写；③ 高频判断："功能在原版测过，本地化版可以免测"——错，本地化版是另一个构建产物。

> **过渡：**六类是"测什么"，接下来看"怎么组织"，业界有两份公开的本地化测试计划可以参照。

【8.24】软件本地化测试（2 分钟）

> 这一页把本地化测试再细分成六块，并且给了两份业界公开的测试计划作参照。先看这六块，它比上一页的六类更贴近"执行分工"。
>
> **I18n 测试**——查国际化能力，源语言版就能做。**L10n 测试**——查本地化版本本身。**语言/翻译的测试**——纯语言层面：漏译、错译、术语、语气。**美观/界面测试**——排版、对齐、截断、控件布局。**功能测试**——业务功能在本地化环境下还对不对。**发布测试**——打包、安装、升级、许可、版本号、语言包分发。
>
> 这六块的划分很有意思：它把"语言"和"美观"分开，把"功能"和"发布"分开。为什么？因为**责任人和修法不同**：语言问题返给译者，美观问题返给 UI/前端，功能问题返给开发，发布问题返给构建和运维。**缺陷分类分对了，流转就快；分错了，来回踢皮球。**
>
> 课件给了两个参考链接：一个是 Mozilla 的 Firefox 2.0 L10n 测试计划，一个是 OpenOffice 的本地化测试计

划。这两个都是开源项目公开的文档，价值在于——你能看到**真实的、工业级的本地化测试计划长什么样**：有哪些测试项、怎么组织多语言、怎么记录每种语言的状态。

我建议你课后真的去翻一翻。看别人成熟的测试计划，比看十页理论都快。看的时候带着三个问题：**它列了哪些测试项？每种语言怎么标记完成度？缺陷怎么归类？** 这三问的答案，就是你自己写计划时的模板。

还有一个观察：这类公开计划里通常有一张「语言状态表」，把每种语言标成未开始、翻译中、待验证、已通过。这张表是本地化项目管理的核心工具，因为多语言并行时，你随时得知道哪一种语言卡在哪一步。没有这张表，项目经理只能凭感觉汇报进度。

板书：六块：I18n 测试｜L10n 测试｜语言/翻译测试｜美观/界面测试｜功能测试｜发布测试。右侧标注责任人：译者｜UI/前端｜开发｜构建运维。底部写参考："Mozilla Firefox L10n Test Plan｜OpenOffice L10n Test Plan"。

提问：一个界面上"德语文案读起来别扭"和"德语按钮文字被截断"，这两条缺陷应该分到哪一块？

预设回答：

- 答"都是翻译问题"——错。前者是语言问题，后者是布局问题，修的人和方法完全不同。

- 答"别扭的给译者，截断的给前端"——完全正确，这就是分类的意义。

- 答"都提给开发，让他们判断"——可以，但效率最低。测试的分类能力，直接决定缺陷流转的速度。

- 收口：所以这一页真正的考点不是六个名词，而是"**分类即分派**"这个意识。

易错点 / 考点：① 六块与上一页六类的区别：这页按"测试类型/责任人"分，上页按"测试内容"分；② 两个参考计划的出处要能说出（Mozilla、OpenOffice）；③ 案例题常给一条缺陷让你分类，注意"语言"与"美观"的界线。

过渡：六块里，前两块最容易混——i18n 测试和 l10n 测试到底差在哪？下一页专门做对比。

【8.25】I18N 测试 vs. L10N 测试（3 分钟）

这一页是对比页，把两个测试放在一起看。我给你一张"五维对照表"，这五个维度就是考试问"两者区别"时的答题骨架。

第一维，测的对象。i18n 测的是"**能力**"——这个软件有没有被本地化的资格；l10n 测的是"**成品**"——这个特定

语言的版本做得好不好。

第二维，什么时候做。i18n 早，源语言版就能做，甚至在设计评审阶段就开始；l10n 晚，必须等翻译交付、重新构建之后。

第三维，语言环境。i18n 通常在**单语言环境**下做——界面还是源语言，只是切换区域、时区、键盘；l10n 必须在**目标语言环境**下做——装目标语言的操作系统、目标语言的输入法，最好在目标国家实测。

第四维，缺陷归属。i18n 的缺陷基本都是**开发**的：硬编码、按字节截断、默认编码写死；l10n 的缺陷可能属于译者（错译、术语不一致）、可能属于 UI（截断、重叠）、也可能属于开发（资源没加载）。**归属不同，返工成本差一个数量级。**

第五维，复用性。i18n 测一次，对所有语言都有效；l10n 每个语言都要测一遍，而且语言越多，总工作量线性增长。

这五维里最关键的是第二维。因为**越早发现问题越便宜**，这是软件工程里唯一一条几乎无例外的规律。所以业界的推荐做法是：把 i18n 测试做扎实，让 l10n 测试专注于语言和排版——前者像体检，后者像装修验收。

最后提醒一句：两者不能互相替代。i18n 全过，l10n 照样可能因为译文质量差而不合格；l10n 全过，也不证明这个软件能顺利支持第十种语言。

> **板书：**五维对照表（两列）：对象（能力/成品）｜时机（早/晚）｜语言环境（单语切区域/目标语言环境）｜缺陷归属（开发/译者・UI・开发）｜复用性（一次通吃/每语言一遍）。底部："i18n 像体检，l10n 像装修验收；不能互相替代"。

提问：一个软件通过了全部 i18n 测试，能不能说它"本地化没问题"？

预设回答：

- 答"能，国际化都过了说明底子好"——错。底子好只说明"能装"，不说明"装得对"。译文质量、排版效果都不在 i18n 的检查范围里。

- 答"不能，还得看具体语言的译文和排版"——对，这正是 l10n 测试的职责。

- 答"不能，还得看目标国家的文化禁忌"——补得好，这属于 l10n 的文化适用性。

- 追问收口：一句话记住——**i18n 决定"能不能"，l10n 决定"好不好"**。

易错点 / 考点：① 五维对照是简答题的标准骨架；② 高频判断："i18n 测试通过即代表本地化质量合格"——

错；③ "单语言环境 vs 多语言环境"这一维最容易被漏，答题时补上能加分。

> **过渡：**8.1 到此结束。进入 8.2，先看翻译验证要验哪三样东西。

【8.26】8.2 翻译验证（2 分钟）

8.2 讲翻译验证。这一节只有三项，但每一项都值得认真对待，因为翻译是本地化里唯一"用户每天都在读"的产出。

第一项，**翻译的内容**。这是基本盘：有没有漏译（源语言有、目标语言没有）、有没有错译（意思歪了）、有没有多译（源语言删了、译文还留着）、术语是否统一、数字和变量占位符有没有被译坏。最后这一条特别技术：句子里如果有个 %s 或者 {0}，译者一动它，程序运行时就会崩或者显示错位。

第二项，**目标语言的文化心理**。这一项最微妙。同一句话，语法全对，但读起来"不像人话"，或者语气不对——比如把面向消费者的轻松语气译成了公文腔，或者把正式场合的敬语译成了口语。还有一种情况是"**政治正确与敏感度**"：某些表达在某个文化里带有冒犯意味，虽然字面没错。这种问题机器查不出来，只能靠目标语言的母语者读。

第三项，**特殊符号**。引号（中文的""和英文的""）、破折号、省略号、货币符号、度量单位符号、数学符号、版权符号。这些符号在不同语言里的形态和用法不一样，翻译时容易原样照抄，结果在目标语言里显得很别扭，甚至显示不出来。

三项合起来是一句话：**翻译验证不是"查错别字"，是判断"这段文字在这个文化里，像不像一个本地人写的"**。这也是为什么本地化行业的黄金标准是"**由目标语言的母语者做审校**"，而不是靠双语词典对照。

> **板书：**三项：① 翻译的内容（漏译・错译・多译・术语・占位符）② 目标语言的文化心理（语气・敬语・敏感度）③ 特殊符号（引号・破折号・货币・单位）。底部："像不像本地人写的"。

> **提问：**译文语法全对、意思也对，但用户说"读起来很别扭"，这算缺陷吗？

> **预设回答：**

- 答"算，属于可读性问题"——算，这类缺陷在本地化项目里很常见，通常定为中低级但必须改。

- 答"不算，意思对就行"——这是典型错答。软件是给人用的，"读着别扭"直接影响信任感，用户会怀疑这软件是不是山寨的。

- 答"要看是哪类产品，正式文档要求高一些"——很专业的补充，产品定位决定语言标准的严格程度。

- 收口：所以验收标准要提前写清楚：是"能看懂"还是"读起来自然"，这两档的返工量差很多。

易错点 / 考点： ① 三项内容必背；② "占位符被译坏"是技术性极强的考点，要能举 %s、{0} 这类例子；③ 高频判断："翻译验证可以由不懂目标语言的人用工具完成"——只能完成一部分（漏译、术语一致性），语气和文化心理必须母语者判断。

【思政落点 3】：翻译这一项最考验"**对用户负责**"这四个字。一段机翻痕迹很重的界面，功能一个不少，可用户第一眼就给它贴了个"不用心"的标签。反过来，一个术语从头到尾统一、语气从头到尾一致的软件，用户会觉得"这家公司是认真的"。做技术的人容易有个错觉：语言是"软"的部分，不如代码重要。可用户接触产品的**唯一通道就是这些文字**。请记住：**你交付的每一个字，都是产品的一部分；对文字敷衍，就是对用户敷衍。**

过渡：翻译只是本地化的一半，另一半是"翻译之外的改动"。下一页看这两层分别改什么。

【8.27】软件本地化与翻译（3 分钟）

这一页把本地化拆成两个层面：**技术层面的更改**和**文化层面的更改**。这个二分法是本章的骨架之一，答题时非常好用。

先说**技术层面**，五项：**调整大小**——控件、对话框、列宽要能容纳译文；**调整默认设置**——默认语言、默认区域、默认纸张、默认货币，都要按目标地区设；**重新编译**——本地化版是重新构建的产物，这句话我重复第三遍了，因为它太重要；**创建新的图形**——图片里带文字的、图标带英文缩写的，都要重做；**重新编排文档格式**——帮助文档、PDF 的页面尺寸、字体、目录、索引都要重排。

这五项的共同点：**都是工程活，都可以量化验收**。控件宽度是不是够、默认值对不对、编译有没有报错、图片文字有没有换、文档页数有没有错乱，一条条都能写成用例。

再说**文化层面**，课件列了：**包装、图标、宣传、样品、政治敏感的术语、地方规章和宗教信仰**。这一层的共同点是：**没有对错，只有合适不合适**。同一个手势，在有些国家是"OK"，在有些国家是侮辱；同一个颜色，在有些文化里代表喜庆，在另一些文化里代表丧事；地图上的边界画法，在有些国家是法律问题。

为什么要把这两层分开？因为**处理方式完全不同**：技术层靠测试和返工，文化层靠**评审和本地顾问**。技术缺陷可以靠用例抓，文化问题必须靠"懂当地的人"提前审。你让一个不懂当地文化的测试团队去测"这个图标在

当地合不合适"，他们测不出来——不是不专业，是信息不够。

所以成熟的做法是：技术层由测试团队兜住，文化层建立一道"**本地审校 + 合规评审**"的关卡，两道关都过，才允许发布。

板书：左右两栏。左"技术层面（可量化、靠测试）"：调整大小｜调整默认设置｜重新编译｜创建新图形｜重排文档格式。右"文化层面（靠评审、靠顾问）"：包装｜图标｜宣传｜样品｜政治敏感术语｜地方规章与宗教信仰。底部："技术靠用例，文化靠人"。

提问：一个"竖起大拇指"的图标，作为"点赞"功能放进产品，需要本地化评审吗？

预设回答：

- 答"需要，有些地区这个手势是冒犯"——对，这就是文化层面的典型例子。

- 答"不需要，这就是个通用图标"——这是最常见的错答。正是这种"想当然的通用"，造成了大量出海事故。

- 答"看目标市场，如果只发国内就无所谓"——合理，文化评审的范围应该跟发布范围挂钩，这是成本意识。

- 追问收口：所以文化层面的检查项，要**按目标市场逐条过**，而不是全球一套标准。

易错点 / 考点：① 两层各自的五项/六项要能分开举例，别把"重新编译"写进文化层；② 高频判断："文化适用性问题是缺陷"——更准确的说法是"适用性/合规问题"，处理路径与功能缺陷不同；③ 简答常问"本地化为什么不能等同于翻译"，就用这两层来答。

【思政落点 4】：文化层面这一栏，考的是"**尊重差异、守住合规**"。有些内容在我们看来稀松平常，在别人的文化里却是禁区；有些表述在国内没问题，到了别的国家就触碰法律。这不是"要不要迁就别人"的问题，而是两条底线：**尊重，是跨文化合作的基本教养；合规，是产品能不能在当地合法存在的前提。** 做技术的同学将来很可能参与出海项目，请把这两句话记住：不确定的地方，去问当地的人；拿不准的内容，按最严格的标准处理。**敬畏规则，才走得更远。**

过渡：技术层面第一件要改的就是"大小"，因为译文一长，界面就装不下。下一页专门看文字扩展。

【8.28】翻译问题 – 文字扩展（2 分钟）

这一页讲**文字扩展**，也就是译文比原文长导致的界面问题。这是本地化测试里出现频率最高、最容易被用户看到的一类缺陷。

先讲为什么会扩展。不同语言的"信息密度"不一样：同一个意思，用不同语言写出来，字符数可以差很多。前面 8.3.2 会提到一句课件原话——**德语最长，汉语比较精炼**。这不是印象，是结构差异：德语喜欢把词组合成很长的复合词，一个词能顶中文一句话；中文是单字表意，同样的信息用更少的字符就能承载。反过来，如果把中文译成英文，多数情况下也会变长，因为英文需要更多的词和冠词介词。

扩展会打到哪些地方？第一，**按钮和标签**——宽度是设计时按源语言定的，一长就截断或者被挤成两行，两行又撑高工具栏。第二，**表格列**——列宽固定，长文字要么截断要么把整张表撑到屏幕外。第三，**对话框和提示**——文字超出边界、按钮被挤出可视区，用户点不到"确定"。第四，**菜单**——菜单项换行、子菜单跑到屏幕外。第五，**帮助文档和错误信息**——段落长度变了，页码、索引、交叉引用全要重排。

测试这一类问题的方法很简单，就两个字：**用最长的语言去试**。业界的做法是先做伪翻译（把文本机械拉长），把布局问题在翻译交付前就炸出来；然后在真实语言里重点测德语、俄语这类"长"的语言。

还有一条更根本的建议：**布局不要按字符数写死**。控件要能自适应、能换行、能滚动；设计时就要给译文留出余量。这一条属于 i18n 的设计标准，也是前面 8.1 第八条"改善翻译文本尺寸，具有调整的灵活性"的落地。

> **板书：**一句原理："德语最长，汉语精炼"；五个受灾点：按钮标签｜表格列｜对话框提示｜菜单｜文档与错误信息。底部两行："用最长的语言去试"｜"布局不按字符数写死"。

提问：界面上一个按钮在德语下文字被切成两行，把按钮撑高了，导致整个工具栏错位，这条缺陷该提给谁？

预设回答：

- 答"提给译者，让他把德语缩短"——有时可行，但治标：换成俄语还会犯，而且为了排版牺牲译文准确性是本末倒置。

- 答"提给前端，让按钮宽度自适应或允许换行"——这是治本的方向。

- 答"提给 UI 设计，因为设计时没留余量"——最根本的一层，也是最有价值的判断。

- 追问收口：所以合理的缺陷单会写"**建议工具栏按钮宽度自适应；短期可由译者提供简短译法作为缓解**"——治本加治标，两条都给。

易错点 / 考点：① "德语最长、汉语最精炼"是课件原话，考试可直接引用；② 文字扩展的五个受灾点是案例题素材；③ 高频判断："译文太长就让译者改短"——只在极少数场景可接受，根本解法是布局自适应。

过渡：8.2 讲完，进入 8.3——本地化测试的技术问题，三块，第一大块是数据格式。

【8.29】8.3 软件本地化测试技术（1 分钟）

8.3 是本章最"能直接用"的一节，三块内容，我先把地图给你。

8.3.1 数据格式——六项：数字、货币、时间、日期格式、度量衡单位、复数问题。这一块的特点是**全部可以用"对照"来测**：同一份数据，在两种区域设置下各显示一次，比对是否符合该地区的习惯。

8.3.2 页面显示和布局——三件事：译文长度差异（德语最长、汉语精炼）、乱码（双字节语言 GB/BIG5/JP）、字符索引与排序。这一块的特点是**看着测**：截图、比对、走查。

8.3.3 配置和兼容性问题——键盘布局、打印机配置、硬件与版本兼容、数据库问题、热键。这一块的特点是**搭环境测**：装目标语言的系统、换键盘、插打印机。

三块合起来，正好对应本地化缺陷的三种"发现方式"：**对照数据、看界面、搭环境**。你以后拿到一个本地化版本的测试任务，就按这三个动作依次做：先对照数据格式，再走查界面布局，最后在目标环境里跑一遍兼容性。

一句话总结这一节的价值：**8.1 告诉你"为什么"，8.3 告诉你"查什么"，8.4 告诉你"怎么组织"。**

> **板书：**三块：8.3.1 数据格式（数字・货币・时间・日期・度量衡・复数）→ 8.3.2 显示与布局（长度・乱码・索引排序）→ 8.3.3 配置与兼容（键盘・打印机・硬件版本・数据库・热键）。右侧写三种发现方式："对照数据｜看界面｜搭环境"。

提问：这三块里，哪一块最可能靠自动化脚本覆盖？

预设回答：

- 答"数据格式，因为规则明确"——对，格式可以写断言，输入一组区域设置，比对输出字符串。
- 答"乱码，可以扫"——部分可以，比如扫资源文件里的非法字节序列，但界面渲染出来的乱码要靠截图比对。
- 答"都能自动化"——过于乐观。布局重叠、文字被遮挡、方向错误，这些机器很难判"好不好看"。
- 收口：所以本地化测试的现实是：**格式类尽量自动化，界面类靠人眼和截图比对**。

易错点 / 考点：① 三块的名称与内容不能混，尤其"字符索引排序"属于 8.3.2 而不是 8.3.1；② 三种发现方式（对照/看/搭环境）是很好用的记忆抓手；③ 简答若问"本地化测试的技术问题有哪些"，就答这三块。

过渡：进入 8.3.1 之前，先看一个几乎每个本地化项目都会踩的坑——姓名。

【8.30】最常见的问题（2 分钟）

这一页拿姓名开刀，因为姓名是"最常见的问题"，也是最能说明本地化思维的一页。

课件给的对照很简洁：中文的"周宇文"、"朱先生"；英文的"Shaomin Zhu"、"Mr. Zhu"。这一组对照里藏着四层差异，我逐层说。

第一层，顺序。中文是姓在前、名在后；英文是名在前、姓在后。所以"周宇文"转成英文写成 Shaomin Zhu 的时候，两个部分的位置换了个个儿。程序如果按"第一个词是姓"去解析，遇到英文名就全错。

第二层，称谓的位置。中文说"朱先生"，称谓在姓**后面**；英文说 Mr. Zhu，称谓在姓**前面**。如果程序用模板拼"某某先生"，换到英文就拼成了"Mr. 放在后面"，非常别扭。

第三层，称谓的选择。中文的"先生/女士"和英文的 Mr./Ms./Mrs. 不是一一对应：英文里 Mrs. 还带婚姻状况信息，而现代用法里很多人更愿意用 Ms.。这一层涉及文化习惯，靠规则表解决不了，只能靠本地化规范。

第四层，也是最要命的一层：排序和检索。通讯录按什么排？中文按拼音排、按笔画排、还是按 Unicode 编码排？英文按姓氏的字母序排。同一个"周"，拼音是 Z，在字母表最后；如果按编码排，位置又完全不同。**用户找一个人的时候，排序错了等于功能不可用。**

所以姓名这个字段，看着是"两个文本框"，实际上牵动四件事：**显示顺序、称谓模板、音译一致性、排序规则**。这就是本地化测试的思维方式——**一个普通字段，也要问一遍"这个规则在别的国家还成立吗"**。

顺便说一句，课件为什么把它叫「最常见的问题」：因为姓名几乎出现在每个系统的第一屏——注册、登录、通讯录、订单、发票、报表。它被使用的次数最多，出错的机会自然也最多。你将来做本地化走查，从姓名入手是性价比最高的起点。

> **板书：**四层：① 顺序（姓+名 / 名+姓）② 称谓位置（朱先生 / Mr. Zhu）③ 称谓选择（先生女士 / Mr. Ms. Mrs.）④ 排序与检索（拼音・笔画・编码 / 字母序）。底部："一个字段，四种规则"。

提问：通讯录里"周宇文"和"Shaomin Zhu"是同一个人，系统该怎么存才不会乱？

预设回答：

- 答"存成一个字符串就行，显示的时候再排"——错，一个字符串没法可靠地拆出姓和名，排序和称谓全做不了。
- 答"分成姓、名、称谓、显示名四个字段"——这是比较标准的做法，显示名单独存，避免每次拼。
- 答"存拼音就够了"——不行，中文用户在界面上要看到汉字，拼音只是排序和检索用的辅助。

- 追问收口：所以姓名设计至少要有"**结构化字段 + 本地显示名 + 排序键**"三部分，这是很多系统吃亏之后才补上的。

> **易错点 / 考点：** ① 姓名的四层差异要能说全，只答"顺序不同"是最常见的失分点；② 高频判断："姓名就是一个文本字段"——错，它牵动显示、称谓、音译、排序；③ 排序规则（拼音/笔画/编码）是常考细节。

> **过渡：** 姓名只是一个字段，8.3.1 要把所有"格式类"字段系统过一遍，六项。

【8.31】8.3.1 数据格式（3 分钟）

这一页是数据格式的六项清单：数字、货币、时间、日期格式、度量衡的单位、复数问题。这六项是本地化测试的"基础套餐"，我一项一项给你判据。

第一，数字。两个点：千分位用什么符号（逗号、空格、点、还是不用），小数点用什么符号（点还是逗号）。这两个符号在有些地区是**互换**的，所以同一串字符在不同地区读出来的数值差一千倍。

第二，货币。符号是什么（¥、$、€）、符号放在数字前面还是后面、小数保留几位、负数怎么表示（减号、括号、还是红色）。还有汇率和金额精度——金额的舍入规则如果按源语言写死，财务报表会对不上。

第三，时间。12 小时制还是 24 小时制、上午下午怎么标、时分隔符是冒号还是点。还有时区：显示本地时间还是服务器时间，跨时区跨夏令时的时候日期会不会跳。

第四，日期格式。年月日的顺序、分隔符、月份是数字还是名称、星期从哪天开始。这就是第 2 页那个 2/3/2009 的问题。

第五，度量衡的单位。长度、重量、面积、温度、速度、纸张尺寸。公里/英里、摄氏度/华氏度、千克/磅、A4/Letter。这一项在工业软件、物流、医疗类系统里是硬需求。

第六，复数问题。这一项最容易被忽略，也最能体现语言差异。英语只有单数和复数两态（one apple / two apples）；有些语言有多达好几种复数形式，不同数量用不同词形；中文、日文基本不分单复数。程序如果硬编码"数量 > 1 就加 s"，在那些语言里就错了。

六项的测试手法是统一的：**准备一组"边界数据"，在两种以上的区域设置下各跑一遍，逐项对照是否符合当地习惯**。边界数据要包括：负数、零、很大很小的数、带小数的金额、跨年的日期、夏令时切换点。

> **板书：** 六行：① 数字（千分位・小数点）② 货币（符号・位置・小数位・负数）③ 时间（12/24・时区・

夏令时）④ 日期（顺序・分隔符・月份・起始星期）⑤ 度量衡（长度・重量・温度・纸张）⑥ 复数（单复数・多复数形态）。底部："边界数据 + 两种区域设置对照"。

提问：为什么"复数问题"会被单列成一项？它不是语法问题吗？

预设回答：

- 答"因为程序里到处是'共 N 条'这种拼接，复数处理错了整句话就错"——对，这是最实际的理由。

- 答"因为有些语言的复数形式不止两种"——非常准确，这是它必须单列的技术原因：不是"加不加 s"，而是"选哪种词形"。

- 答"复数跟软件测试没关系"——错。凡是"数量 + 单位"的界面文字，都涉及复数规则，这在电商、报表、消息提示里到处都是。

- 追问收口：所以正确做法是把复数规则交给本地化框架去处理，而不是在代码里写 if (n > 1)。

易错点 / 考点：① 六项要能默写并各举一例；② "小数点与千分位互换"是最经典的案例，务必能讲清后果；③ 高频判断："复数就是加 s"——错，多复数形态语言里这是错的。

过渡：这六项都跟"区域设置"有关，下一页看区域设置在哪里配、影响什么。

【8.32】区域与语言（2 分钟）

这一页是区域与语言设置的例子，也就是操作系统里那个"你住在哪儿、你说什么语言"的配置界面。别小看这两个下拉框，本地化测试的大部分现象都由它决定。

它一般分三块，你要知道每块管什么。

第一块，**格式**。它决定日期怎么写、时间用不用 24 小时、数字的千分位和小数点、货币符号、度量衡单位。这一块是纯显示层的，改它不影响程序逻辑，只影响呈现——这正是 8.3.1 那六项的实验台。

第二块，**位置**。它决定你所在的国家/地区，进而影响一周从哪天开始、默认纸张、默认度量衡。有些系统还把"位置"用于首页内容、天气、时区推断。

第三块，**管理里的"非 Unicode 程序的语言"**。这一项最关键，也最容易被忽略。它的作用是告诉系统："那些不支持 Unicode 的老程序，用哪套编码去解释它们的文字。"改这一项通常要重启。**很多"某个老软件打开中文全是乱码"的问题，根子就在这里**——老程序用的是系统默认代码页，而默认代码页跟你的文件编码不一

致。

测试上怎么用这一页？我建议你做三组实验：**只改格式不改语言**（看格式类显示变不变）、**只改语言不改格式**（看界面文字变不变）、**改非 Unicode 程序的语言**（看老模块会不会乱码）。三组实验分别对应显示、翻译、编码三个层次，做完你对这个软件的区域处理能力就有底了。

还有一个坑要提醒：**改完设置要重启或者重新登录才生效，而且不同版本的 Windows 表现不一样**。测试时如果没重启就下结论，很容易误判成缺陷。

> **板书：**三块：格式（日期・时间・数字・货币・度量衡）｜位置（国家地区・一周起点・纸张）｜管理→非 Unicode 程序语言（代码页，需重启）。底部三组实验："只改格式｜只改语言｜改默认代码页"。

提问："某个老模块打开中文全是乱码，其它模块正常"，你最该先查什么？

预设回答：

- 答"查那个模块是不是不支持 Unicode"——方向对，这类老模块通常按系统默认代码页解码。

- 答"查系统的非 Unicode 程序语言设置"——非常准确，这就是最常见的原因。

- 答"重装字体"——错。字体解决字形，不解决解码。

- 收口：所以这类问题的修复往往有两种：改代码支持 Unicode（治本），或调整系统默认代码页（缓解，且影响其他程序）。

易错点 / 考点：① 格式/位置/非 Unicode 语言三块各管什么，别混；② "改设置需重启才生效"是测试执行细节，常被忽略导致误判；③ 高频考点：非 Unicode 程序语言与代码页的关系。

过渡：配置讲完，我们看这些格式在真实界面上出错的样子。

【8.33】本地化问题 - 数据格式（2 分钟）

这一页是数据格式出错的实例。我把常见的错误形态归成五类，你以后看到就能立刻认出来。

第一类，符号错位。比如该用逗号做千分位的地方用了点，或者小数点用了逗号。这类错误在数字上最危险，因为它不报错，只是数值含义变了。

第二类，顺序颠倒。日期写成月/日/年但目标地区习惯日/月/年；姓名写成名在前但目标地区习惯姓在前。这

类错误用户能看出来，但会显得很不专业。

第三类，单位没换。该显示公里显示英里，该显示摄氏度显示华氏度，纸张该用 A4 用了 Letter。这类错误在打印和报表上后果最明显——直接印错。

第四类，格式串硬编码。整块界面上只有某几个字段的格式不对，其它都对。这通常是那几个字段用了自己拼的字符串，没走统一的格式化函数。**这一类是最值得提缺陷的，因为它暴露的是架构问题**。

第五类，时区处理错误。显示的是服务器时间、跨时区时日期差一天、夏令时切换那天的记录时间错乱。这类缺陷特别隐蔽，只有在特定日期、特定时区才能复现。

怎么测这五类？核心手法还是"**对照**"：同一份数据，在两种区域设置下各截一张图，把两边的格式逐字比对。比对的时候不要凭记忆，要拿一张"目标地区的正确格式"参照表——这张表可以从目标地区的官方习惯、或者系统自带的格式里取。

最后一句：这一类缺陷**最容易在评审时被当成"小问题"放过**，但它们在金融、医疗、物流系统里是会出真事故的。判严重度的时候，看它有没有流进数据层。

> **板书：**五类：符号错位｜顺序颠倒｜单位没换｜格式串硬编码｜时区处理错误。右侧写测试手法："同一数据・两种区域・逐字比对"。

提问：界面上只有两三个字段的日期格式不对，其它都对，你判断根因是什么？

预设回答：

- 答"这两个字段是特殊处理的"——对，通常就是有人图省事自己拼了字符串。

- 答"可能是翻译错了"——不太可能。日期格式不是翻译出来的，是格式化函数算出来的。

- 答"可能是这两个字段的数据类型不对，存的是字符串"——非常专业，这是根因层面的判断：字符串就没法格式化。

- 追问收口：所以这条缺陷的价值不在"改这两个字段"，而在"**全项目搜一遍还有多少地方在拼格式字符串**"。

易错点 / 考点：① 五类错误形态要能认出来；② "局部格式不对"往往指向硬编码，这是案例题的高分点；③ 严重度判断标准：是否流入数据层、是否影响金额或时间。

过渡：数据格式说完，进入 8.3.2——页面显示和布局，第一件事就是文本长度。

【8.34】8.3.2 页面显示和布局（3 分钟）

这一页是显示与布局的总纲，三句话，每句都能展开成一堆测试点。

第一句，**德语最长，汉语比较精炼**。这句话是本地化布局问题的总根源。德语靠复合词承载信息，一个词能很长；中文一字一义，同样的意思用的字符少。所以把中文界面译成德语，几乎必然出现"装不下"；反过来把英文译成中文，往往会出现"留白太多"。测试的含义是：**做布局测试，一定要用"最长的语言"和"最短的语言"各测一遍**，中间的语言自然会过。

第二句，**乱码（双字节语言 GB/BIG5/JP/...）**。课件括号里列的这三个，代表的是三类典型的双字节编码：GB 是简体中文的国标，BIG5 是繁体中文，JP 是日文。它们的共同点是"一个字符用两个字节"，也共同面临"编码不一致就乱码"的风险。这一句还提示了一个现实：**多语言环境里往往同时存在好几种编码**，一个系统里既有 UTF-8 的数据，又有历史遗留的 GB 数据，还有从日文系统导出的文件——这才是真实项目的复杂度。

第三句，**字符索引、排序**。索引指按什么建目录：按拼音、按笔画、按部首、按字母。排序指按什么规则比大小：不同语言的字母表顺序不同，还有变音符号、大小写、数字混排的处理规则。这一项在中文环境里就已经很复杂（拼音序和笔画序结果不同），多语言下更复杂——一个列表里同时有中文、英文、日文，按什么排？

三句话归一句：**布局测试要问三件事——长度够不够、编码通不通、顺序对不对。** 这三件事分别对应三类工具：截图比对、编码扫描、排序结果核对。

> **板书：**三句：① 德语最长・汉语精炼（用最长最短各测一遍）② 乱码（GB / BIG5 / JP 双字节）③ 字符索引与排序（拼音・笔画・部首・字母序）。右侧："长度够不够｜编码通不通｜顺序对不对"。

提问：一个列表里同时有中文、英文、日文的名字，你打算按什么规则排序？

预设回答：

- 答"统一按 Unicode 编码排"——技术上可行、结果稳定，但用户看不懂这个顺序（汉字按编码排既不是拼音也不是笔画）。

- 答"按当前界面语言的规则排"——这是主流做法，中文界面按拼音、英文界面按字母，比较符合用户预期。

- 答"随便排，用户会自己搜"——错。排序是检索的基础功能，排序乱意味着用户找不到东西。

- 追问收口：所以排序规则要**明确写进需求**，并且写清"混合语言时怎么办"，否则开发和测试各按各的理解做，

必吵。

> **易错点 / 考点：**① "德语最长、汉语精炼"是课件原话；② GB/BIG5/JP 分别对应简体、繁体、日文，别搞混；③ 索引（拼音/笔画/部首）与排序（字母序/编码序）是两个概念，考试常混。

> **过渡：**装不下的时候，界面上具体是什么样子？下一页给的是"不能完全显示"。

【8.35】示例：不能完全显示（2 分钟）

这一页是"不能完全显示"的实例。这是本地化缺陷里用户最容易发现、也最影响观感的一类。我把它的四种典型形态说清楚，你以后看图就能对号入座。

第一种，硬截断。文字被直接切掉，末尾没有任何提示。用户看到的是半句话，而且不知道后面还有内容。这是最差的一种。

第二种，加省略号。文字被截断但末尾有省略号，用户知道"还有内容"，但看不到。这种比硬截断好，前提是有悬浮提示或者能点开看全文；如果没有，用户就只能猜。

第三种，撑破容器。文字没被截断，但把控件撑变形了——按钮变成长条、对话框被顶宽、工具栏换行导致下面所有内容下移。这种问题往往连带一片：一个控件变宽，整行布局全乱。

第四种，压盖。文字盖住了旁边的图标或其它文字，两层内容叠在一起，两边都读不了。

这四种形态的严重度不一样：**压盖和撑破最严重**（因为影响其它元素），硬截断次之，加省略号最轻（前提是能看全文）。

测试的时候要注意一个细节：**同一个界面，在不同分辨率、不同字号、不同 DPI 下，表现可能完全不同**。你在 1920 宽屏上看着正常，用户在 1366 的笔记本上就截断了。所以本地化布局测试一定要**在目标地区的常见分辨率下测**，而不是只在自己的开发机上瞄一眼。

还有一条经验：截图时要**同时截源语言版和目标语言版的同一个界面**，两张并排放进缺陷单。开发一看就知道差多少，不用来回问。

> **板书：**四形态：硬截断｜省略号｜撑破容器｜压盖。标严重度：压盖・撑破 > 硬截断 > 省略号。右侧："多分辨率・多字号・多 DPI 都要试"。

> **提问：**界面文字加了省略号，算不算合格？

预设回答：

- 答"算，至少有提示"——不完全算。要看能不能看到全文：悬停提示、点开详情、或者能拖动列宽。看不到全文，用户还是没法用。

- 答"不算，应该让控件自适应"——这是治本的要求，但现实中列宽固定是常态，省略号加完整提示是常见的折中方案。

- 答"看是什么字段，关键信息不能省"——非常专业。金额、日期、编号这类字段被截断，后果远比描述性文字严重。

- 收口：所以判合格的标准是：**信息能不能被用户完整获取**，而不是"看起来有没有提示"。

易错点 / 考点： ① 四形态与严重度排序；② "同一界面在不同分辨率/DPI 下表现不同"是执行要点，常被漏；③ 缺陷单要"双语截图并排"，这是实操规范。

过渡： 装不下是一种问题，读不出来是另一种——下一页看乱码。

【8.36】示例：乱码（2 分钟）

这一页是乱码的实例。前面 8.1.2 讲过原理，这一页看现象，并且教你**从现象反推原因**——这是测试人员最值钱的一项技能。

看现象分三种。

第一种，成串的怪字符。原来的中文位置变成了一堆莫名其妙的符号，比如"锟斤拷"、"Ã¤Â¸Â"这样的东西。这种的根因几乎总是**编码不一致**：数据是 UTF-8，程序按 GBK 解了；或者数据是 GBK，程序按 UTF-8 解了。特征是同一条数据在别的机器、别的入口能正常显示。

第二种，方框或者豆腐块。每个字的位置是一个空心方框，或者一个带问号的方块。这不是解码错，是**字体缺字形**：编码解对了，但这个字体里没有这个字符的形状。特征是换个字体就正常。

第三种，半个字或者问号。一个汉字显示成半个，或者干脆变成"?"。这通常是**截断造成字节断裂**，或者经过了一次"无法表示就替换成问号"的有损转换。特征是这个字段往往有长度限制。

三种现象，三种根因，三种修法：**第一种改编码声明或转换逻辑，第二种装字体或换字体，第三种改截断方式（按字符截，不按字节截）**。这就是为什么我反复强调"先定位层次，再提缺陷"——你报"界面有乱码"，开

发只能猜；你说"该字段在 UTF-8 数据源下按 GBK 解码，建议统一为 UTF-8"，开发十分钟就能改完。

补一条经验：乱码问题有一个特点——**它往往不是全乱，而是局部乱**。因为只有经过某条特定路径的数据才会错。所以复现时要记清"**数据从哪来、经过哪些环节、在哪个环节被转码**"，这条链路就是根因所在。

板书：三现象三根因：成串怪字符 → 编码不一致｜方框豆腐块 → 字体缺字形｜半个字或问号 → 字节截断/有损替换。右侧："先定位层次，再提缺陷"；底部："记清数据链路：来源 → 环节 → 转码点"。

提问：同一段文字，在网页上正常，导出成文件再用记事本打开就乱码，你怎么解释？

预设回答：

- 答"文件编码和网页编码不一样"——对，网页有编码声明（响应头或页面里的声明），文件没有，记事本只能猜。

- 答"记事本太老了"——这是事实层面的补充：老版本记事本对编码的自动识别很弱，容易猜错。

- 答"文件坏了"——错。文件本身完好，是"解释方式"错了。

- 追问收口：所以导出功能要保证"**文件里带编码声明**"，或者统一用兼容性最好的编码。这一条可以直接写成一条测试用例。

易错点 / 考点：① 三种现象与三种根因的对应关系必须能背；② "局部乱码"的复现要记录数据链路，这是实操要点；③ 高频判断："乱码装个字体就好"——只对第二种现象成立。

过渡：显示问题还有一类不在文字上，而在图片和控件里——下一页叫"其它"。

【8.37】示例：其它（2 分钟）

这一页的标题叫"其它"，其实是把所有不属于"文字长度"和"乱码"的界面本地化问题收在一起。我按四类给你补全，因为这一类在实际项目里漏得最多。

第一类，图片里的文字。图标上写着 "Save"，按钮背景图里印着 "Submit"，截图里带着英文的界面元素。这类文字**抽不进资源文件**，只能重画图。测试时要专门去找"图片里的文字"，包括图标、徽标、示意图、帮助文档里的截图——这是最容易被漏的一类，因为测试人员的注意力都在文本上。

第二类，图标的隐喻。邮筒、手势、动物、颜色、节日元素。同一个图标在不同文化里含义可能完全不同。这类问题不属于"显示错误"，属于"文化不适"。

第三类，隐含的格式假设。比如地址栏按"省-市-区-街道"排；比如姓名字段假设"两个字就是姓名"；比如"邮编"字段只留 6 位数字。这些假设在源语言环境里天经地义，换个国家就全崩。测试时要把每个输入框都问一遍"**这个格式在别的国家还成立吗**"。

第四类，与语言绑定的行为。大小写转换（有些语言没有大小写）、首字母大写（有些语言不这么用）、连字符断词（只对拉丁语言有效）、文字方向、以及"按字母分组"的联系人列表——中文该按拼音分组，日文按假名，这些都算行为层面的本地化。

这一页的教学目的，是让你建立"**本地化不只是文本**"的意识。文本只占界面的一部分，图片、图标、布局、行为假设，都是本地化的工作面。你在做本地化走查时，眼睛要能"分层看"：文字一层、图片一层、控件一层、行为一层。

板书：四类：图片里的文字（需重画）｜图标隐喻（文化）｜隐含格式假设（地址・邮编・姓名）｜与语言绑定的行为（大小写・断词・方向・分组）。右侧："本地化不只是文本，分层看：文字｜图片｜控件｜行为"。

提问：一个"按首字母分组"的联系人列表，在中文环境下该怎么改？

预设回答：

- 答"改成按拼音首字母分组"——这是主流做法，中文用户也熟悉。

- 答"改成按笔画或者部首"——可行，但用户认知成本高，实际产品很少这么做。

- 答"不用改，中文也能按编码分组"——错。按编码分组出来的分组名对用户毫无意义。

- 追问收口：所以这类"与语言绑定的行为"，改动量往往比翻译大得多——它要改逻辑，不只是改文本。

易错点 / 考点：① "图片里的文字抽不出来，必须重画"是高频考点；② 隐含格式假设的四类例子（地址、姓名、邮编、电话）要能举；③ 常考判断："本地化就是把文字翻译一遍"——这一页就是反驳它的四条证据。

过渡：把上面这些零散的点系统化，就是下一页的 UI 验证细节，六条。

【8.38】UI 验证的细节（4 分钟）

这一页是本章最实用的一页，六条 UI 验证细节，每条都能直接变成检查项。讲完给你一张走查表。

第一条，控件相互重叠或排列间隔不均衡。重叠是硬伤——两个控件压在一起，用户点不准；间隔不均衡

是"看着别扭"——有的地方挤、有的地方空。根因通常是布局用了绝对坐标，译文一变长就挤到别人地盘上。

第二条，文字遮挡图像、文字超过边界、控件中字符没有完整显示。这三种是"装不下"的三个变体，但要注意区分：遮挡是层级问题（z-order），超边界是容器问题，字符不完整是裁剪问题。修法不同。

第三条，文字方向的问题，比如希伯莱文和阿拉伯文是从右到左显示。这里课件点出了两种 RTL 语言：希伯莱文和阿拉伯文。这一条要验的是：整段方向对不对、控件顺序有没有镜像、标点位置对不对、数字在 RTL 段落里的位置对不对。

第四条，左右对齐问题。课件给了两个很好的例子：阿拉伯文应右对齐；中英文之间有区别——**中文段落开头要空两个字，英文开头不空**。这个细节特别能说明问题：首行缩进不是"美观偏好"，是**排版规则**，按语言不同而不同。如果程序统一加了缩进，英文段落就会显得很怪；反之中文段落不缩进，读起来也不符合习惯。

第五条，连字符对多数拉丁语言有效，但对东方语言一般无效。连字符断词是为了在行末把长单词拆开，拉丁语言按音节拆是合理的；中文、日文没有"词间空格"，也不需要连字符断词，中文可以在任意字之间换行。所以如果排版引擎给中文加了连字符，那就是错的。

第六条，拉丁语言的大小写问题、多字节语言的显示乱码问题。大小写：某些界面会做"全部大写"的样式处理，遇到没有大小写的语言（中文、日文、阿拉伯文）会怎样？乱码：还是前面讲的那三类现象。

六条走查完，它们覆盖的正是 UI 本地化"看得见"的全部：**位置、边界、方向、对齐、断行、字符**。口诀：**"挤不挤、装不装、朝哪边、对不对齐、怎么断、显不显"**。

> **板书：**六条：① 控件重叠・间隔不均 ② 文字遮挡图像・超边界・字符不完整 ③ 文字方向（希伯莱・阿拉伯 RTL）④ 左右对齐（阿拉伯右对齐；中文首行空两字，英文不空）⑤ 连字符对拉丁有效、东方无效 ⑥ 大小写・多字节乱码。底部口诀："挤不挤・装不装・朝哪边・对不对齐・怎么断・显不显"。

提问：中文段落首行空两个字、英文不空，这个差异该由谁保证——翻译、设计，还是代码？

预设回答：

- 答"翻译的时候在每段前面加两个空格"——这是最常见的土办法，但很脆弱：换字号、换字体、复制粘贴都会错位。

- 答"由排版/样式控制，按语言设置首行缩进"——这是正确做法，样式跟着语言走。

- 答"无所谓，看不出来"——错。排版规则是本地化的基本礼貌，专业用户一眼就能看出"这不是本地做的"。

- 追问收口：所以样式也要国际化——**同一个样式类，在不同语言下可以有不同的缩进、行高、字体**。

易错点 / 考点： ① 六条要能按"位置・边界・方向・对齐・断行・字符"归类；② "中文首行缩进两字、英文不缩进"是课件明确给的细节，非常适合出选择题；③ 高频判断："连字符断词对所有语言都适用"——错，对东方语言一般无效。

过渡： 界面看完了，还有一层在"环境"里——配置与兼容性。

【8.39】8.3.3 配置和兼容性问题（3 分钟）

8.3.3 讲配置和兼容性。这一块的特点是：**问题不在代码里，在环境里**。同一份安装包，装在不同配置的机器上表现不同。课件给了三句话，我逐句展开。

第一句，**配置性包括键盘布局设计，它是语言依赖性最大的硬件，还有打印机配置等**。注意"语言依赖性最大的硬件"这个说法——键盘。为什么？因为键盘是物理的：德语键盘的 Z 和 Y 位置对调，法语键盘是 AZERTY 布局，阿拉伯语键盘上还有专门的键。软件如果按物理位置或者按字符绑快捷键，换个键盘就可能失效或者串键。打印机配置也是配置的一部分：纸张尺寸、默认边距、驱动语言，都会影响输出。

第二句，**兼容性包括与硬件的兼容性、与上一版本的数据兼容及与其他本地化软件的兼容性等等**。三类兼容要分清：**硬件兼容**（在目标地区的常见机型上能不能跑）、**版本兼容**（从上一版升级上来，数据和设置会不会坏）、**与其它本地化软件的兼容**（同机装了别的语言包、别的输入法、别的办公套件，会不会冲突）。第三类最容易被忽略，但现实中最常见——用户机器上永远不止你一个软件。

第三句，**关注一些点，如数据库问题、热键**。数据库问题指的是：数据库的字符集、排序规则、字段长度，跟本地化数据能不能对上。一个 GBK 排序规则的库，存进 UTF-8 的数据，轻则乱码重则截断。热键前面讲过：菜单助记符要重新分配，还要避开系统保留键和其它软件的占用。

三句话合起来：**配置看键盘和打印机，兼容看硬件版本和邻居软件，细节看数据库和热键**。这一块没有巧办法，只能**搭环境、真机跑**。

板书： 三句：① 配置性（键盘布局＝语言依赖最大的硬件・打印机配置）② 兼容性（硬件・上一版本数据・其它本地化软件）③ 关注点（数据库・热键）。底部："配置靠搭环境，兼容靠真机跑"。

提问： 为什么课件说键盘是"语言依赖性最大的硬件"？

预设回答：

- 答"因为不同语言的键盘布局不一样，键位和字符的对应关系不同"——对，这是核心原因。

- 答"因为键盘上有当地语言的字母"——也算一层：阿拉伯语、俄语键盘上印的是当地字母。

- 答"键盘就是输入设备，跟语言没关系"——错。同一台电脑插上不同布局的键盘，敲同一个物理键出来的字符可能完全不同。

- 追问收口：所以测"快捷键"时，不能只在自己的键盘上按一遍，要**在目标语言的键盘布局下按一遍**。

> **易错点 / 考点：**① "键盘是语言依赖性最大的硬件"是课件原话，常出填空或判断；② 兼容性三类（硬件・版本数据・其它本地化软件）要能说全；③ 数据库的字符集与排序规则是高频细节考点。

> **过渡：**配置和兼容性具体怎么测？下一页给了三个非常具体的场景。

【8.40】配置和兼容性测试（2 分钟）

这一页给的是配置与兼容性测试的三个具体切入点，加上一个参考资料。这三个点都很"工程"，我把它们讲透。

第一，本地化操作系统。测本地化版本，最好在**目标语言的操作系统**上测，而不是在自己机器上装个语言包。为什么？因为语言包只改界面语言，系统的默认代码页、区域格式、字体集合、输入法、甚至某些 API 的行为可能都没变。你在中文系统上装个德语语言包，测出来的"通过"是不可靠的。这一条是所有本地化测试规范里的第一条。

第二，Non-Unicode 数据路径。指的是程序处理"非 Unicode 编码的数据"时走的那些路径：读一个 GBK 的文本文件、导入一个 BIG5 的 CSV、访问一个老数据库。这类路径在现代代码里往往是历史遗留，最容易出乱码和截断。测试方法很直接：**准备一批不同编码的数据文件，逐个导入，看结果**。

第三，欧洲系统本地化 OEM vs. Windows "ANSI"。这一条讲的是编码生态的复杂性：同一个 Windows，在不同地区的 OEM 版本里，默认的"ANSI"代码页可能不一样——西欧是 1252，中欧是 1250，中文是 936，日文是 932。所谓"ANSI"其实是各家代码页的统称，不是一种编码。所以一个程序如果依赖"系统 ANSI 代码页"，它在不同地区的机器上行为就不同。这正是"**同一份程序，在不同国家的机器上表现不一样**"的技术根源。

课件还给了一篇 W3C 的文章，标题是 "Migrating to Unicode"——迁移到 Unicode。这篇文章值得看，它讲的是"**怎么把一个老的单字节/本地编码系统，安全地迁到 Unicode**"，包括数据迁移、代码改造、以及迁移过程中怎么避免数据损坏。这类文章对理解 8.3.3 的全部问题非常有帮助。

三个点归一句：**本地化测试必须在"真环境"里做，而真环境的核心差异，一半在操作系统，一半在编码。**

> **板书：**三点：① 本地化操作系统（不是装语言包）② Non-Unicode 数据路径（老编码文件・老库）③ 欧洲 OEM vs Windows "ANSI"（代码页 1252/1250/936/932）。右侧写参考资料："W3C — Migrating to Unicode"。底部："真环境＝操作系统 + 编码"。

提问：在中文 Windows 上装德语语言包，能不能替代德语版 Windows 做测试?

预设回答：

- 答"能，界面都变德语了"——错。界面语言只是表层，代码页、字体、输入法、区域默认值都可能没变。

- 答"不能，但可以作为初筛"——这是很务实的判断：语言包环境能发现一部分翻译和布局问题，但编码类问题必须上真系统。

- 答"看测试目的，测翻译够用，测编码不够"——完全正确，这就是分层测试的思路。

- 收口：所以测试计划里要写清"**哪一类用例在语言包环境跑，哪一类必须上目标语言系统**"，这是资源有限时的现实取舍。

易错点 / 考点：① "ANSI 不是一种编码，是一堆代码页的统称"这一点极易考；② 三个切入点的名称要能说出；③ 高频判断："装语言包等于本地化环境"——错。

过渡：8.3 的技术问题到此结束，进入 8.4——把本地化版本当成一个完整软件来测。

【8.41】8.4 本地化功能测试（1 分钟）

8.4 只有一页标题，但它标志着一个转折：前面 8.3 讲的是"**一类一类地查问题**"，8.4 讲的是"**把本地化版本当成一个完整软件，从头到尾测一遍**"。

这两件事不一样。8.3 是清单式的：查格式、查布局、查兼容。8.4 是场景式的：一个用户拿到德语版，从安装、登录、用功能、打印、导出、看帮助、升级，整条链路走下来，会不会在某一步卡住。

为什么必须做这一步？因为清单再全，也覆盖不了"组合"。举个典型场景：日期格式在列表页显示正常（8.3.1 过了），但把这个日期导出成 Excel 再导入回来，就错了——这个缺陷既不在格式清单里，也不在布局清单里，它藏在"**导出导入这条链路**"上。只有按业务流程走一遍，才会撞见。

所以 8.4 的关键词是"**集成**"：不只是界面翻译对了，而是整条业务链在目标语言环境下跑得通。

下一页我们看这一节的具体工作怎么分。

板书：中间一条横线：左边"8.3 = 清单式（查一类一类问题）"，右边"8.4 = 场景式（按业务链路走一遍）"。下方写："清单覆盖不了组合，组合缺陷只在链路里出现"。

提问：清单式测试和场景式测试，哪个更容易漏缺陷？漏的是哪一类？

预设回答：

- 答"清单容易漏，因为清单是有限的"——对，清单只能覆盖"想到的"。

- 答"场景容易漏，因为没有明确判据"——也有道理，场景测试依赖测试人员的经验和想象力。

- 答"两种都会漏，但漏的东西不一样：清单漏组合，场景漏细节"——这个回答最完整。

- 收口：所以成熟的测试策略是**两者叠加**：先按清单扫一遍保证覆盖率，再按场景走一遍保证连通性。

易错点 / 考点：① 8.3 与 8.4 的定位差别（清单 vs 场景）是理解本章结构的关键；② 高频判断："本地化测试就是检查界面翻译"——8.3 和 8.4 都在反驳它；③ 案例题常给"导出再导入出错"这类跨环节缺陷，考点就是"组合缺陷"。

过渡：进入具体工作，本地化功能测试分成三块，还对应两种行业通行的组织形式。

【8.42】本地化功能测试的具体工作（3 分钟）

这一页把本地化测试分成三块——**语言、功能、美观/界面**——然后给出两种组织形式：**In-country testing（本土测试）**和 **Translation verification testing（TVT，翻译验证测试）**。这一页信息密度很高，我分两半讲。

先讲三块。**语言**：查译文本身，漏译、错译、术语、语气、符号。**功能**：查业务功能在目标语言环境下是否正常，包括输入、处理、输出、打印、导出。**美观/界面**：查排版、对齐、截断、方向、图标。

这三块的划分很实用，因为它直接对应三类执行者：语言由译审或本地语言专家做，功能由测试工程师做，美观由 UI/前端或有审美判断的人做。**分对了人，效率就上来了。**

再讲两种组织形式。

In-country testing，本土测试，也叫"在当地测"。意思是把测试活动放到目标市场去做：用当地的机器、当

地的网络、当地的系统版本、当地的用户习惯，最好由当地的人来测。它的价值在于"**真实**"：你在国内模拟不出来的东西——当地的网速、当地的支付习惯、当地常见的机型、当地用户的操作直觉——只有在当地才能撞见。代价是贵、协调难。

Translation verification testing，TVT，翻译验证测试。它的关注点窄而深：**只验翻译**。典型做法是语言专家拿着源语言和目标语言对照的界面，逐条检查译文是否准确、是否符合语境、术语是否统一。TVT 通常有专门的工具支持：双语对照界面、术语库比对、上下文截图。

两种形式的关系：**TVT 保证"字对"，In-country testing 保证"用得对"**。预算有限时，先用 TVT 把语言问题清干净（便宜、可并行），再用本土测试做最终把关（贵、但不可替代）。

板书：三块：语言｜功能｜美观/界面（对应译审・测试・UI）。两种组织形式：In-country testing（本土测试・真实环境・当地用户）｜TVT（翻译验证测试・双语对照・只验翻译）。底部："TVT 保字对，本土测试保用得对"。

提问：TVT 和普通的"翻译审校"有什么区别?

预设回答：

- 答"TVT 是在软件界面上看，审校是看文档"——很准确。TVT 的关键在于"**在上下文里看**"：同一句话在按钮上和在提示框里，译法可能不同。

- 答"TVT 要检查术语一致性，还要看有没有截断"——补充得好，TVT 往往连带发现布局问题。

- 答"没区别，都是看翻译"——错。脱离界面的审校看不到语境、看不到长度限制、也发现不了占位符错误。

- 收口：所以 TVT 是"**在真实界面里验翻译**"，这是它不可替代的价值。

易错点 / 考点：① 三块内容与三种执行者的对应关系；② In-country testing 与 TVT 的英文全称、中文译名、定位差别都要能说；③ 高频判断："TVT 可以替代本土测试"——错，两者互补。

过渡：这一节最后落到 L10N 功能测试的具体内容：集成、索引排序、联机文档。

【8.43】L10N 功能测试（3 分钟）

这一页是 L10N 功能测试的三项内容，加一段很重要的说明，我把说明放在最后讲，因为它最容易被忽略。

第一项，**集成测试**。本地化版本要和什么集成？和操作系统、和数据库、和打印机、和外部系统接口、和其

它语言版本共存。典型的集成问题是：装了两个语言版本互相覆盖、语言包和主程序版本不匹配、和本地的办公软件交换文件时编码错乱。这一项的本质是"**本地化版本不是孤岛**"。

第二项，**索引和排序**。前面 8.3.2 提过原理，这里是功能层的验证：搜索能不能搜到、按字母/拼音/笔画建的索引对不对、排序结果符不符合当地习惯、分页和"跳到某个字母"能不能用。这一项在通讯录、文档管理、电商商品分类里是核心功能，排序错了等于搜索废了。

第三项，**联机文档的功能测试**。注意是"功能"不是"内容"：帮助能不能打开、目录能不能跳转、搜索能不能用、索引链接有没有失效、PDF 里的书签和字体正不正常。文档的内容准确性属于翻译验证，文档的**可用性**属于功能测试，这两件事要分开提缺陷。

然后是那段说明，我认为是全章最实用的一句：**不仅要查看用户界面，而且要对文件保存、打印等类似的功能进行测试，特别要注意语言环境特定的组件，比如时间、日期格式以及文字处理等相关方面的功能进行测试。**

这句话在纠什么偏？纠的是"本地化测试就是看界面"这个偏见。真正的本地化缺陷，一大半在"**界面之外**"：保存下来的文件名对不对、导出文件的编码对不对、打印出来的纸张和边距对不对、复制粘贴到别的软件里会不会乱码。这些都是"看不见界面"的地方，却恰恰是用户真正在干活的地方。

所以这一页我给你一句行动指令：**测本地化版本时，把"保存、打开、打印、导出、复制粘贴、发邮件"这六个动作，每个都用目标语言的数据做一遍。**

> **板书：**三项：集成测试｜索引和排序｜联机文档功能测试。下方框内写："界面之外：保存・打开・打印・导出・复制粘贴・发邮件——每个都要用目标语言数据做一遍"。右侧批注："语言环境特定组件：时间・日期・文字处理"。

提问：为什么"打印"在本地化测试里要被单独强调？

预设回答：

- 答"因为纸张尺寸不一样，A4 和 Letter 会错位"——对，这是最直接的原因。

- 答"因为打印出来的字体可能不对，中文可能变方框"——也对，打印链路用的是另一套字体和渲染。

- 答"打印跟本地化没关系"——错。打印涉及纸张、边距、字体、编码、驱动，四样都可能跟地区有关。

- 追问收口：所以打印机这一环，是"界面看着好、输出却错"的典型场景，也是 8.3.3 里"打印机配置"被列出来的原因。

易错点 / 考点： ① 三项内容要能说出；② 那段说明是全章的高频引用句，"界面之外的功能"是核心观点；③ 高频判断："本地化测试只需要看界面"——这一页就是标准反例。

过渡： 本章的内容到这里讲完了，我们收一下，把整章压缩成三句话。

【8.44】本章小结（3 分钟）

这一页是本章小结，三句话，正好对应本章的三层结构。我按"研发、测试、执行"的顺序给你串一遍。

第一句，**软件国际化是研发上多工程问题，在设计、编程上需要遵守 I18N 相应的规范和要求**。注意"多工程"这个说法——它不是说"很多工程"，而是强调国际化是**贯穿设计、编码、构建、文档的系统性工程**，不是一个可以后期补上的功能。这一句对应 8.1：i18n 属 Engineering，做一次管所有语言。

第二句，**国际化测试需要验证双向识别功能、语言切换方式、多字节和单字节文字的混合、输入法编辑器、大小写转换、快捷组合键、词序问题、无硬编码**。这一串就是 8.1 那十项测试点的浓缩版。注意最后三个字"**无硬编码**"——它是整句的落脚点，因为硬编码是绝大多数国际化缺陷的共同根因。

第三句，**本地化测试会涉及"数据格式、UI 显示和布局、配置和兼容性问题、翻译内容等"**。这一句对应 8.3 和 8.4：四类问题，加上前面 8.1.5 的六类测试工作、8.2 的翻译验证，构成完整的执行面。

三句话合起来，就是一张责任图：**研发负责"能做到"，测试负责"验证做到了"，本地化团队负责"做得好"。**

最后我给你一个"三问自检"，考试和面试都用得上。拿到任何本地化任务，先问三句：

一、**这个功能有没有把地区相关的东西硬编码？**（查 i18n 根子）

二、**同一份数据在目标地区显示成什么样？**（查格式与布局）

三、**在目标环境里，整条业务链跑得通吗？**（查集成与兼容）

三问都过了，本地化质量就基本靠得住了。

板书： 三句：① 国际化＝研发的多工程问题（设计・编程守 I18N 规范）② 国际化测试八项（双向・切换・混排・IME・大小写・快捷键・词序・无硬编码）③ 本地化测试四类（数据格式・显示布局・配置兼容・翻译内容）。底部三问："硬编码了吗？显示成什么样？整条链跑得通吗？"

提问： 如果只允许你保留一条本地化测试用例，你保留哪一条？为什么？

预设回答：

- 答"切换语言后逐页走查"——覆盖面最广，一条用例能把翻译、布局、方向、编码都过一遍。

- 答"用德语在德语系统上跑一遍主流程"——很有价值，一条用例同时压到长度、格式、环境。

- 答"随便留一条，反正测不完"——这种心态要纠正。测试资源永远不够，**取舍能力才是专业能力**：留哪条，取决于这个产品最怕哪类事故。

- 收口：所以真正专业的回答是"**看业务风险**"——金融产品留格式和金额，出海消费产品留布局和翻译。

易错点 / 考点： ① 三句话要能背，尤其"无硬编码"和"多工程"这两个词；② 本章的四组数字容易混：8.1 的八条能力、五条标准、八步流程、六类测试，答题前先对号；③ 高频综合题："结合本章，谈谈如何组织一个多语言版本的测试"——就按"i18n 先行 → 伪翻译 → TVT → 本土测试 → 发布验证"这条线答。

过渡：最后留几分钟，把可以继续深挖的资料给你们。

【8.45】参考资源（1 分钟）

这一页是本章的参考资源，五个入口，我按"用处"给你排个序，你课后按需取用。

第一个，**W3C 的 International 站点**。这是国际化领域最权威的公共资源之一，讲字符编码、语言标签、文字方向、本地化最佳实践。8.1.2 和 8.3.3 那些标准，很多能在这里找到面向开发者的解释。**如果你只想看一个地方，就看它。**

第二个，**Sakai 项目的 I18N 主页**。这是一个开源教学系统项目的国际化资料，价值在于"**工程实践**"：他们怎么抽资源、怎么管理多语言、怎么处理翻译流程。看真实项目怎么做，比看规范更接地气。

第三个，**GALA Global**，全球化与本地化行业协会。这里能看到本地化这个行业的全貌：产业分工、质量标准、术语、职业路径。对想了解"本地化是一门生意"的同学特别有用。

第四个，**MSDN 上 Kieran 的博客**。这是一个长期写国际化与编码问题的技术博客，特点是"**具体**"：一个乱码案例能写清来龙去脉。这类博客最适合在遇到实际问题时去翻。

第五个，**Java 官方的国际化入门文章**。讲 Java 里怎么做国际化：资源包、区域设置、格式化类。如果你要做实验，从这里入手最快——Java 的国际化框架是最经典的实现之一，看懂了它，其它语言的框架都是同一套思路。

五个资源，用法是：**W3C 打底、Sakai 看流程、GALA 看行业、博客查案例、Java 文章做实验**。别指望一次看完，遇到具体问题时去查，效率最高；而且看的时候记得把「它讲的是原理、流程还是案例」分清楚，别拿一篇博客当标准，也别拿一份标准当操作手册。

> **板书：**五个入口：W3C /International｜Sakai I18N｜GALA Global｜MSDN 博客｜Java 国际化入门。右侧写用法："W3C 打底・Sakai 看流程・GALA 看行业・博客查案例・Java 做实验"。

提问：如果你要给团队写一份"本地化测试入门材料"，你会先看这里的哪一个？

预设回答：

- 答"W3C，最权威"——权威但偏原理，写入门材料可能太硬。

- 答"Sakai，因为是完整项目的实践"——很实用的选择，能直接看到测试计划和组织方式。

- 答"都不用看，我自己总结"——不建议。本地化是个成熟行业，别人踩过的坑已经写成文档了，重踩一遍很浪费。

- 收口：写材料的最佳组合是"**一份原理（W3C）+ 一份实践（Sakai）+ 一份自己的项目案例**"。

易错点 / 考点：① 五个资源各自的性质要能对上（标准组织・开源项目・行业协会・技术博客・语言框架文档）；② 考试一般不考网址，但可能考"本章推荐了哪几类参考资料"；③ 记住 GALA 是行业协会，别和标准组织混。

过渡：这一章就讲到这里，最后说两句收尾的话。

【8.46】感 谢 聆 听（1 分钟）

第 8 章讲完了，最后我用三句话把这一章收住。

第一句，**本地化测试的难点不在技术，在"换位"**。你要能站到一个德国用户、一个阿拉伯语用户、一个美国用户的位置上，去问"这样写我看不看得懂、用不用得顺"。这个能力，比会写脚本更难得，也更值钱。

第二句，**越早做越便宜**。国际化是设计阶段的事，本地化是发布前的事。你在设计评审时多问一句"这个格式硬编码了吗"，可能省掉后面几周的返工。这是软件工程里少数几条几乎无例外的规律。

第三句，**细节就是专业**。一个编码参数、一个首行缩进、一个称谓位置、一个复数词形——每一件都不起眼，但用户看到的全部，就是这些不起眼的东西。

> 从下一章开始，我们换一个视角，从"测一个软件"转到"管一个测试项目"。这一章希望大家带走的不只是知识点，还有一种习惯：**看到一个理所当然的设计，先问一句"换个地方还成立吗"**。
>
> 谢谢大家。

> **板书：**三句：① 难点在换位，不在技术 ② 越早做越便宜 ③ 细节就是专业。底部留一行："换个地方还成立吗？"

> **提问：**这一章里，哪一个知识点最可能在你将来的工作中直接用到？

> **预设回答：**

- 答"字符集和乱码的排查思路"——很实用，几乎每个项目都会遇到。

- 答"文字扩展和布局自适应"——也很实用，做过出海项目的人都会认同。

- 答"这些都用不上，我们只做国内市场"——不一定。国内也有繁体版、少数民族语言版、多终端多分辨率，方法是一样的。

- 收口：所以这一章的方法论，其实是"**如何在差异中保证一致性**"，这个能力在哪儿都用得上。

> **易错点 / 考点：**① 本章的三条主线（i18n 能力・l10n 执行・文化合规）要能在脑子里连成一张图；② 复习时优先背四组数字：八条能力、五条标准、八步流程、六类测试；③ 案例题的通用答法：**先判层次（i18n 还是 l10n）→ 再判类型（格式/布局/编码/文化）→ 最后给建议**。

> **过渡：**本章结束，下一章进入测试项目管理，我们从"怎么测"走向"怎么管"。

四、思政落点清单（共 4 处）

序号	所在页	落点主题
1	8.11	字符集这一页最值得琢磨的地方是——它出错的时候，程序一切正常。没有异常、没有崩溃
2	8.21	这八步里最不起眼的是第一步——配置管理。它不产出任何看得见的东西，却决定了后面七
3	8.26	翻译这一项最考验"对用户负责"这四个字。一段机翻痕迹很重的界面，功能一个不少，可
4	8.27	文化层面这一栏，考的是"尊重差异、守住合规"。有些内容在我们看来稀松平常，在别人