Slide 1: Slide 2: : Slide 3:

Slide 4: Spark 발전 과정

여러분 이 차트를 보세요. 2009년부터 2015년까지 Spark의 발전 과정입니다. Spark도 계속 성장하고 있습니다. 2009년에 Spark는 아직 "아기"였는데, 2015년에는 이미 "빅스타"가 되었습니다! 이 상승하는 추세 그래프는 우리에게 Spark를 배우기로 선택하는 것이 절대 현명한 결정이라는 것을 말해줍니다.

Slide 5: Spark 아키텍처

이것은 Spark 가족의 전체 가족사진입니다! Spark Core는 마치 가족의 아버지 같아서, 가장 중요한 책임을 담당합니다. 위에는 네 명의 자녀가 살고 있습니다:

- Spark SQL
- Spark Streaming
- MLlib
- GraphX

각 자녀는 모두 자신만의 특기가 있습니다. 오른쪽의 표시 "RDD 애플리케이션 개발 완료"를 주목하세요. 이것이 바로 Spark Core의 주요 직책입니다.

Slide 1: Slide 2: : Slide 3:

Slide 4: Spark发展历程

大家看这个图表，
Spark从2009年到2015年的发展历程。
Spark也在不断成长。
2009年Spark还是个"小baby"，
到了2015年已经变成了"大明星"！

这个上升的趋势图告诉我们，
选择学习Spark绝对是明智的决定。

Slide 5: Spark架构图

这是Spark的家族全家福！
Spark Core就像是家庭的父亲，
承担(chéng dān)着最重要的责任(zérèn)。

上面住着(zhùzhe)四个孩子：

- Spark SQL、
- Spark Streaming、
- MLlib
- 和GraphX。

每个孩子都有自己的特长。

注意右边的标注"完成RDD应用开发"，
这就是Spark Core的主要职责(zhízé)。

Slide 6: Spark SQL 기능

첫 번째 자녀인 Spark SQL을 살펴봅시다.
그것의 초능력은 "관계형 데이터 분석"입니다.
마치 데이터 탐정처럼, 구조화된 데이터를 신속하게 분석하여 데이터 간의 관계를 찾아낼 수 있습니다.

Slide 7: Spark Streaming

두 번째 자녀는 Spark Streaming인데,
그것의 특기는 "스트림 계산 수행"입니다.
상상해 보세요. 데이터는 마치 멈추지 않고 흐르는 강물 같고, Spark Streaming은 이 강에서 가장 뛰어난 선장으로서 실시간으로 흐르는 데이터를 처리할 수 있습니다.

Slide 8: MLlib

세 번째 자녀 MLlib는 "머신러닝 알고리즘 라이브러리"입니다.
그것은 마치 매우 똑똑한 도서관 사서 같아서, 각종 다양한 머신러닝 알고리즘을 소장하고 있습니다.

어떤 알고리즘이 필요하든, 즉시 찾을 수 있습니다!

Slide 6: Spark SQL功能

来看看第一个孩子——Spark SQL。
它的超能力是"分析关系数据"。
就像一个数据侦探(zhēntàn)，
能够快速分析结构化数据，
找出数据之间的关系。

Slide 7: Spark Streaming

第二个孩子是Spark Streaming，
它的特长(tècháng)是"进行流计算"。
想象一下，
数据就像一条永不停歇(yǒng bù tíng xiē)的河流，而Spark Streaming
就是这条河上最厉害的船长(chuánzhǎng)，
能够实时处理流动的数据。

Slide 8: MLlib

第三个孩子MLlib是"机器学习算法库"。

它就像一个超级聪明的图书管理员，
收藏(shōucáng)了
各种各样的机器学习算法。

需要什么算法，
它都能立刻找到！

Slide 9: GraphX

마지막 자녀 GraphX는 전문적으로 "그래프 계산 애플리케이션 프로그램 작성"을 합니다.
그것은 마치 네트워크 전문가 같아서, 소셜 네트워크, 교통 네트워크 등과 같은 복잡한 네트워크 관계를 처리하는 데 특히 뛰어납니다.

Slide 10: Spark 총평

반짝반짝 빛나는 엄지척을 보셨나요?
이것이 바로 우리가 Spark에 대해 평가하는 것입니다 - 최고입니다!
Spark는 마치 빛나는 별처럼 빅데이터의 하늘에서 빛을 발하고 있습니다.

Slide 11: Spark 생태계

이 그림은 Spark의 강력한 생태계를 보여줍니다. 중심에는 Spark가 있고, 주변에는 각종 애플리케이션, 환경 및 데이터 소스가 둘러싸고 있습니다.
마치 번화한 도시처럼, Spark는 도심이고 주변은 모두 각종 다양한 건물과 시설입니다.
왼쪽 상단에 표시된 "Open Source Ecosystem"(오픈소스 생태계)를 주목하세요. 이는 Spark가 오픈소스라는 의미이며, 누구나 사용하고 기여할 수 있습니다!

Slide 9: GraphX

最后一个孩子GraphX专门"编写图计算应用程序"。

它就像一个网络专家，特别擅长(shàncháng)处理复杂的网络关系，
比如社交网络、交通网络等等。

Slide 10:Spark总结

看到这个大拇指和闪闪发光的效果了吗？
这就是我们对Spark的评价——棒极了！
Spark就像一颗闪亮的明星，
在大数据的天空中发光发热。

Slide 11:Spark生态系统

这张图展示了Spark强大的生态系统。
中心是Spark，
周围环绕着 (huánrào zhe)各种应用、环境和数据源。
就像一个繁华的城市，Spark是市中心，
周围都是各种各样的建筑和设施。

注意(zhùyì)左上角标注(shàngjiǎo biāozhù)的是"Open Source Ecosystem"（开源生态系统），
这意味着Spark是开源的，
人人都可以使用和贡献(gòngxiàn)！

Slide 12: Hadoop VS Spark 타이틀

이제 흥미진진한 "대결"을 시작합니다!
Hadoop VS Spark, 마치 두 무술 고수가 겨루기를 하는 것 같습니다.
누가 승리할까요? 함께 살펴봅시다!

Slide 13: MapReduce 문제

MapReduce 的主要缺点 슬라이드 3가지를 말한다.

Slide 14: 높은 지연 문제

여기서 간단한 차트로 MapReduce의 높은 지연 문제를 보여줍니다.
두 작업 간에 데이터 교환이 필요한데,
이것은 마치 두 사람이 대화하지만 중간에 벽이 가로막고 있어서, 쪽지를 전달하여 소통해야 하니 당연히 매우 느립니다!
I/O 오버헤드와 관련이 있습니다.

Slide 15: MapReduce 처리 흐름

이 그림은 MapReduce의 처리 흐름을 자세히 보여줍니다.
Map1, Map2, Map3이 병렬로 처리되고,
그 다음 중간 결과를 출력하며, 모든 Map이 완료되기를 기다린 후에야 Reduce1, Reduce2, Reduce3가 작업을 시작할 수 있습니다.
마치 릴레이 경주처럼, 앞의 선수가 달리기를 끝내지 않으면 뒤의 선수는 시작할 수 없습니다.

Slide 12:Hadoop VS Spark标题

现在我们要进行一场激动人心的"对决"！
Hadoop VS Spark，
就像两个武功高手要进行一场比武。
谁会获胜(huòshèng)呢？
让我们一起来看看！

Slide 13:

MapReduce 的主要缺点 슬라이드 3가지를 말한다.

Slide 14: 延迟高问题

这里用一个简单的图表展示了MapReduce延迟高的问题。
两个任务之间需要进行数据交换，
这就像两个人说话，
但是中间隔着(gézhe)一堵墙(yī dǔ qiáng):
需要通过传纸条来交流，当然会很慢！
传纸条 (chuán zhǐtiáo): "쪽지 전달하다"
涉及IO开销

Slide 15:MapReduce处理流程

这张图详细展示了MapReduce的处理流程。
Map1、Map2、Map3并行处理，
然后输出中间结果，
等待所有Map完成后，
Reduce1、Reduce2、Reduce3才能开始工作。

就像一个接力赛，前面的选手不跑完，
后面的选手就不能开始。

Slide 16: 지연 문제 보충

여기서 높은 지연 문제를 더욱 강조합니다.
여러 작업이 실행 중일 때, 마치 버블티를 사기 위해 줄을 서는 것처럼, 앞사람이 사지 않으면 뒷사람은 계속 기다려야 합니다.

Slide 17: Hadoop MapReduce 실행 흐름 상세도

이것은 더 자세한 Hadoop MapReduce 실행 흐름도입니다.
우리는 데이터가 HDFS에서 읽혀지고, 처리를 거쳐, 다시 HDFS에 쓰여진 다음, 다음 반복에서 이 과정을 다시 반복하는 것을 봅니다.
마치 이사하는 것처럼, 매번 모든 물건을 옮겨야 하니 매우 피곤합니다!

Slide 16:第16页：延迟问题补充

这里进一步强调了延迟高的问题。

多个任务在执行中，
就像排队买奶茶一样，
前面的人不买完，
后面的人就得一直等着。

Slide 17:Hadoop MapReduce 执行流程详图(xiángtú)

这是一个更详细的Hadoop MapReduce 执行流程图。
我们看到数据需要从HDFS读取，经过处理，再写入HDFS，然后下一个迭代再重复这个过程。
就像搬家一样，
每次都要把所有东西搬来搬去，
很累人！

Slide 18: Spark의 장점

이제 "신예" Spark의 우수성을 살펴봅시다!
그것은 네 가지 주요 장점이 있습니다:

1. 다양한 데이터셋 연산 타입
2. 더 유연한 프로그래밍 모델
3. 메모리 계산 제공
4. DAG 기반 작업 스케줄링 실행 메커니즘

마치 만능형 슈퍼 히어로 같습니다!

Slide 19: Hadoop과 Spark 실행 흐름 비교

이것은 매우 중요한 비교 그림입니다!
위는 Hadoop MapReduce의 실행 흐름으로, 매 반복마다 디스크에 쓰여집니다.
아래는 Spark의 실행 흐름으로, 중간 결과가 메모리에 저장될 수 있습니다.

Slide 18: Spark的优点

现在我们来看看Spark这位"新星"的优势！
它有四大优点：

1. 多种数据集操作类型、
2. 编程模型更灵活、
3. 提供了内存计算、
4. 基于DAG的任务调度执行机制。

就像一个全能型的超级英雄！

Slide 19: Hadoop与Spark执行流程对比

这是一个非常重要的对比图！
上面是Hadoop MapReduce的执行流程，
每次迭代都要写入磁盘；

下面是Spark的执行流程，
中间结果可以存储在内存中。

Slide 20: 성능 비교 실험

이 놀라운 비교 결과를 보세요!
로지스틱 회귀 알고리즘을 실행할 때,
Hadoop은 110초가 필요하지만 Spark는
단지 0.9초만 필요합니다!
이것은 마치 거북이와 토끼의 경주
같은데, 이번에는 토끼가 게으르지 않고
거북이보다 100배 이상 빠르게
달렸습니다!

Slide 21: 문제 1 - Spark가 Hadoop을 대체할까요?

이것은 많은 학생들이 묻는 질문입니다.
이 귀여운 아이콘과 당황한 사람을 보면,
여러분 지금 표정과 비슷하지 않나요?
이 질문에 답해봅시다.

Slide 22:

우리는 Hadoop의 구성을 분석했습니다.
Hadoop은 HDFS와 MapReduce 두 부분을
포함합니다.
Spark는 주로 MapReduce를 대체하러
왔지, 전체 Hadoop을 대체하는 것이
아닙니다.
마치 자동차의 엔진을 교체하는 것이지,
자동차 전체를 교체하는 것이 아닙니다!

Slide 20: 性能对比实验

看这个令人震惊(zhènjìng)的对比结果！
执行逻辑(luójí sīdì) 回归算法， Hadoop需要110秒， 而Spark只需要0.9秒！

这就像乌龟(wūguī)和兔子(tùzi)赛跑(sàipǎo)
但这次兔子没有偷懒(tōulǎn)，
反而跑得比乌龟快了100多倍！

Slide 21: 问题1 - Spark会取代Hadoop吗？

这是一个很多同学都会问的问题。
看这个可爱的图标和疑惑的小人，
是不是很像你们现在的表情？
让我们来解答这个问题。

Slide 22:

我们分析了Hadoop的组成。
Hadoop包括HDFS和MapReduce两部分。

Spark主要是来取代MapReduce，
而不是整个Hadoop。

就像换汽车的发动机，
不是换整辆车(zhěng liàng chē)！

Slide 23: Spark vs Hadoop 시스템 다이어그램
그림에서 Spark가 대체하는 것은 Hadoop의 MapReduce이며, 전체 Hadoop 시스템이 아닙니다.

Slide 24:
Haddop에서 저장부분 즉 HDFS와 Hbase는 그대로 두고 계산부분인 MapReduce를 Spark로 대체하는 것입니다.

Slide 25: 문제 2 - Spark 개발에는 어떤 프로그래밍 언어를 사용해야 할까요?

또 다른 고전적인 질문입니다! 사람들이 다시 고민하기 시작했습니다.
프로그래밍 언어의 선택은 무기를 선택하는 것만큼 중요합니다.

Slide 26:
Spark는 네 가지 프로그래밍 언어를 지원합니다: **Scala, Python, Java, R**

그들은 마치 네 명의 좋은 친구 같아서, 각자의 특징이 있습니다.

Scala 위에 빨간 하트가 있는 것을 주목하세요. 이는 그것이 Spark의 "최애"라는 것을 의미합니다!

Slide 23: Spark vs Hadoop 系统图
图示中 Spark 替代的是 Hadoop 的 MapReduce，而不是整个 Hadoop 系统。

Slide 24: Hadoop组成分析
在Hadoop中，保留存储部分（即HDFS和HBase），仅将计算部分（MapReduce）替换为Spark。

Slide 25:问题2 - 开发Spark应该使用什么编程语言？

又一个经典问题！

小人们又开始疑惑了。

编程语言的选择就像选择武器一样重要。

Slide 26:Spark支持的编程语言
Spark支持四种编程语言：
Scala、Python、Java、R。

它们就像四个好朋友，各有各的特点。

注意Scala上面有一个红心，
说明它是Spark的"最爱"！

Slide 27: Scala의 우수성

왜 Scala가 첫 번째 선택일까요?

왜냐하면 "Scala는 최고의 성능을 얻을 수 있기 때문입니다!"
마치 가장 적합한 도구를 선택하여 가장 적합한 작업을 하는 것과 같습니다.

Slide 27: Scala的优势

为什么Scala是首选(shǒuxuǎn)呢？

因为"Scala可以获得(huò dé)最好的性能！

就像选择最适合的工具做最合适的工作。

Slide 28: Java와 Python의 단점 정리

이제 Java와 Python의 문제를 구체적으로 분석해봅시다.
여러분 이 컴퓨터 화면을 보세요. 위에 두 가지 주요 문제가 명확하게 나열되어 있습니다:
"Java 코드가 너무 복잡하다"와 "Python 언어의 동시성 성능이 좋지 않다"
마치 교통수단을 선택하는 것처럼, 모두 목적지에 도달할 수 있지만 효율은 완전히 다릅니다!
Java는 마치 구식 완행 기차처럼, 달릴 수는 있지만 너무 느리고 복잡합니다.

Python은 마치 아름다운 스포츠카처럼, 보기에는 좋지만 고속도로(동시 처리)에서는 힘이 부족합니다.

Slide 29: Java 코드 복잡성 문제

여기 매우 흥미로운 비교가 있습니다!
같은 작업에 대해 Scala는 단지 1줄의 코드만 필요하지만, Java는 10줄의 코드가 필요합니다!

이것은 마치 작문을 쓰는 것처럼, 어떤 사람은 한 문장으로 의미를 명확히 표현할 수 있지만, 어떤 사람은 긴 단락을 써야 합니다.
상상해 보세요. 만약 여러분이 매일 10배의 코드를 작성해야 한다면, 죽도록 피곤하지 않을까요?
그래서 적절한 프로그래밍 언어를 선택하는 것이 정말 중요합니다!

Slide 28: Java和Python的缺点总结

现在我们来具体分析一下Java和Python的问题。
大家看这个电脑屏幕，上面清楚地列出了两个主要问题：

Java代码太繁琐(fánsuǒ)和Python语言并发性能不好。
就像选择交通工具一样，
虽然都能到达目的地，
但是效率完全不同！
Java就像老式的绿皮火车，
虽然能跑，但是太慢太复杂；

Python就像一辆漂亮的跑车，
看起来很好，
但是在高速公路上（并发处理）时动力不足。

Slide 29: Java代码繁琐问题

这里有个很有趣的对比！
同样一个任务，
Scala只需要1行代码，
而Java却需要10行代码！
这就像写作文一样，
有些人能用一句话表达清楚的意思，
有些人却需要写一大段(duàn)。

想象一下，如果你每天都要写10倍的代码，是不是会累死？

所以说，选择合适的编程语言真的很重要！

Slide 30: 대화형 프로그래밍 지원 비교

여기서 우리는 중요한 차이점을 봅니다: Scala는 대화형 프로그래밍을 지원하지만, Java는 지원하지 않습니다.

그림에서 Scala에는 녹색 체크 표시 ✓가 있고, Java에는 빨간색 X ✗가 있습니다.

대화형 프로그래밍은 마치 컴퓨터와 채팅하는 것처럼, 한 문장을 물으면 한 문장을 답하며, 디버깅과 학습에 매우 편리합니다.

Java는 마치 매우 진지한 선생님처럼, 전체 작문을 다 써야만 첨삭해 줍니다.

반면 Scala는 마치 친근한 친구처럼, 한 문장을 쓰면 바로 피드백을 줍니다.

Slide 30: 交互式编程支持对比

这里我们看到一个重要的区别：

Scala支持交互式编程，而Java不支持。

大家看图中Scala有一个绿色的对勾duìgōu✓，

Java有一个红色的叉chā✗。

交互式编程就像和电脑聊天一样，

你问一句，它答一句，非常方便调试和学习。

Java就像一个很严肃(yánsù)的老师，

你必须把整篇作文写完才能给你批改(pīgǎi)；

而Scala就像一个亲切的朋友，

你写一句它就能给你反馈。

Slide 31-32: Java와 Python 문제 재강조

우리는 이 두 가지 주요 문제를 다시 강조합니다.
마치 의사가 병 상태를 진단하는 것처럼, 우리는 증상을 반복적으로 확인해야 합니다.

Slide 32: 并发性能对比

여기서 우리는 동시성 성능 비교를 중점적으로 봅니다. Scala는 왕관 👑을 쓰고 있어서, 동시성 성능 면에서 "왕" 수준임을 나타냅니다!
Python도 매우 좋지만, 동시 처리 측면에서는 다소 힘에 부칩니다.
마치 두 운동선수를 비교하는 것처럼, 한 명은 단거리 챔피언이고 한 명은 장거리 선수여서, 다른 경기 종목에서 성능이 완전히 다릅니다.

Slide 33: Scala의 정적 타입 우수성

Scala는 두 가지 중요한 특징이 있습니다: 정적 타입과 대규모 빅데이터 프로젝트 개발에 용이함.
정적 타입은 마치 코드에 방호복을 입힌 것처럼, 컴파일 시 오류를 발견할 수 있어서 실행 시까지 기다릴 필요가 없습니다.
이것은 대규모 프로젝트에 매우 중요하며, 마치 초고층 빌딩을 건설하려면 정확한 설계 도면이 필요한 것과 같습니다.

- **动态类型 (dòngtài lèixíng)**: 동적 타입 (Dynamic Typing)

Slide 31: 重申Java和Python的问题

我们再次强调这两个关键问题。
就像医生诊断病情一样，我们要反复确认症状。
Java的繁琐和Python的并发性能问题是选择编程语言时必须考虑的重要因素。

Slide 32: 并发性能对比

这里我们重点看并发性能(**bìngfā xìngnén)** 的对比。
Scala戴着一个王冠(**wáng guān)** 👑 ，
说明它在并发性能方面是"国王"级别的！

Python虽然也很不错，
但在并发处理方面就显得有些力不从心了。
就像比较两个运动员，
一个是短跑冠军，
一个是长跑选手，
在不同的比赛项目上表现会完全不同。

Slide 33: Scala的静态类型优势

Scala有两个重要特点：
静态**(jìngtài)**类型和便于开发大型大数据项目。
静态类型就像给代码穿上了防护服(**fáng hù fú)**，能在编译时就发现错误，
而不是等到运行时才发现问题。
这对于大型项目来说非常重要，
就像建造摩天大楼需要精确的设计图纸一样。

Slide 34: Scala와 Java의 호환성

여기는 Scala와 Java의 호환성을 보여줍니다.
두 원 중간에 "호환"이라고 쓰여 있어서, Scala가 Java와 잘 협력할 수 있음을 의미합니다.
마치 여러 언어를 구사하는 통역사처럼, 중국어도 할 수 있고 영어도 할 수 있으며, 두 언어 간에 자유롭게 전환할 수 있습니다.
이런 호환성으로 개발자는 Java 생태계의 자원을 충분히 활용할 수 있습니다.

Slide 35: JVM 실행 환경

이 그림은 Scala, Java, Hadoop이 모두 JVM(Java 가상 머신)에서 실행됨을 보여줍니다.
JVM은 마치 만능 운영체제처럼, Java 기반의 모든 언어에 실행 환경을 제공합니다.
Hadoop도 Java로 개발되었기 때문에, Scala는 이 환경에서 마치 물고기가 물을 만난 것처럼 자연스럽습니다.

Slide 34: Scala与Java的兼容性

这里显示了Scala与Java的兼容性。
两个圆圈中间写着"兼容(**jiān róng**)"，

这意味着Scala可以很好地与Java协作。
就像一个会说多种语言的翻译官，
既能说中文，也能说英文，还能在两种语言之间自由切换(**qiē huàn)**。
这种兼容性让开发者可以充分利用Java生态系统的资源。

Slide 35: JVM运行环境

这张图展示了Scala、Java和Hadoop都在JVM（Java虚拟机）上运行。
JVM就像一个万能的操作系统，
为所有基于Java的语言提供运行环境。
Hadoop也是用Java开发的，
所以Scala在这个环境中就像如鱼得水一样自然。

Slide 36: Python과 Hadoop의 관계

여기서 우리는 Python과 Hadoop 간의 관계를 봅니다.

그림은 Python이 일부 복잡한 방식을 통해 Hadoop과 상호작용해야 함을 보여줍니다.

마치 다른 언어를 사용하는 두 사람이 번역을 통해 소통해야 하는 것처럼, 중간에 일부 손실과 복잡성이 있습니다.

Slide 37:

Python의 적용 시나리오를 명확히 지적합니다:

- 기업급 애플리케이션 ❌ (권장하지 않음)
- 과학 연구 시 ✓ (권장)

마치 다른 도구가 다른 장소에 적합한 것처럼, Python은 과학 연구와 프로토타입 개발에 더 적합하며, 대규모 기업급 애플리케이션에는 적합하지 않습니다.

이것은 Python이 나쁘다는 것이 아니라, 적절한 곳에 적절한 도구를 사용해야 한다는 것입니다.

Slide 36: Python与Hadoop的关系

这里我们看到Python和Hadoop之间的关系。

图中显示了Python需要通过一些复杂的方式与Hadoop进行交互，

就像两个说不同语言的人需要通过翻译才能交流一样，

中间会有一些损耗(**sǔnhào)**和复杂性

Slide 37: Python的适用场景

这里明确指出了Python的适用场景：

- 企业级应用❌（不推荐），
- 科研的时候✓（推荐）。

就像不同的工具适合不同的场合，

Python更适合科研和原型开发，

而不太适合大规模的企业级应用。

这不是说Python不好，而是要在合适的地方用合适的工具

Slide 38: Scala는 교육의 첫 번째 선택

마지막으로 우리는 결론을 내립니다:
Scala는 교육의 첫 번째 선택입니다!
마치 교과서를 선택하는 것처럼, 우리는 학습에 가장 적합한 도구를 선택해야 합니다.
Scala는 강력한 기능을 가지고 있으면서도 상대적으로 배우기 쉽고, 최고의 성능도 얻을 수 있어 Spark를 배우는 최선의 선택입니다.
빨간색 Scala 원이 특히 두드러지는데, 마치 무대 위의 주인공을 비추는 조명과 같습니다!

Slide 39: Hadoop 구성 분석

이제 우리는 고전적인 분석 그림을 봅니다.
이 귀여운 Hadoop 코끼리가 두 개의 주요 구성 요소로 분해됩니다:
MapReduce와 HDFS.
MapReduce 옆에는 빨간색 ❌가 있는데, 이것은 무엇을 의미할까요?
이것은 MapReduce 구성 요소에 개선이 필요한 부분이 있다는 것을 의미하며,
HDFS 옆에는 녹색 ✓가 있어서 HDFS는 여전히 매우 좋다는 것을 나타냅니다.
마치 오래된 컴퓨터를 분해하는 것처럼, 어떤 부품은 여전히 사용할 수 있지만 어떤 것은 교체가 필요합니다.

Slide 38: Scala是教学首选

最后，我们得出结论：Scala是教学首选！
就像选择教科书一样，
我们要选择最适合学习的工具。
Scala既有强大的功能，又相对容易学习，还能获得最好的性能，
是学习Spark的最佳选择。
红色的Scala圆圈(yuán quān)特别突出，
就像聚光灯(jùguāngdēng) 照在
舞台上的主角 (zhǔjué)身上一样！

Slide 39: Hadoop组成分析

现在我们看到了一个经典的分析图。
Hadoop这只可爱的小象被分解成两个主要组件：
MapReduce和HDFS。
MapReduce旁边有一个红色的❌，
这意味着什么呢？
这说明MapReduce这个组件有一些需要改进的地方，
而HDFS旁边有一个绿色的 勾号 (gōuhào)✓，
说明HDFS还是很不错的。
就像拆解一台旧电脑，有些零件**língjiàn**还能用，有些就需要更换了。

Slide 40: MapReduce 표현력 제한

여기서 우리는 MapReduce의 첫 번째 문제를 심층 분석합니다: 표현력이 제한적입니다.

이 그림을 보면, 복잡한 비즈니스 로직은 Map 함수와 Reduce 함수로만 표현될 수 있는데, 이것은 마치 가위와 풀로 모든 수공예품을 만드는 것처럼, 도구가 너무 적습니다!

오른쪽에 무한대 기호 ∞가 있는데 사선으로 지워져 있어서, MapReduce가 "분산 애플리케이션 개발의 복잡성을 낮췄지만" 동시에 표현력도 제한했음을 나타냅니다.

Slide 41:

그림은 문제가 어디에 있는지 더 명확하게 보여줍니다. 오른쪽에 주황색 반원이 있고 위에 "표현력 제한"이라고 쓰여 있습니다. 마치 화가에게 흑백 두 색만 제공하는 것처럼, 작품을 그릴 수는 있지만 표현력은 크게 떨어집니다.

복잡한 비즈니스 로직이 Map과 Reduce 두 개의 간단한 함수로 강제로 압축되는데, 이것은 확실히 큰 제한입니다.

Slide 40: MapReduce表达能力有限

这里我们深入分析MapReduce的第一个问题：

表达能力有限。

看这个图，复杂的业务逻辑只能通过Map函数和Reduce函数来表达，

这就像用剪刀和胶水(jiāo shuǐ)做所有的手工作品一样，工具太少了！

右边有一个很形象的无限符号∞，

被一个斜线(xiéxiàn)划掉(huàdiào)了，

说明MapReduce"降低了分布式应用开发的复杂性"，但同时也限制了表达能力。

Slide 41: 表达能力限制详解

这张图更清楚地展示了问题所在。

右边有一个橙色的半圆，上面写着"限制了表达能力"。

就像给画家只提供黑白两种颜色，

虽然也能画出作品，

但是表现力就大打折扣了。

复杂的业务逻辑被强制压缩(**qiángzhì yāsuō)**到Map和Reduce两个简单的函数中，这确实是一个很大的限制(**xiànzhì)**。

Slide 42: 디스크 I/O 오버헤드가 큼

이제 두 번째 문제를 봅시다: 디스크 I/O 오버헤드가 큽니다.
그림은 MapReduce가 중간 결과를 디스크에 써야 한다는 것을 보여주며, 그 하드 디스크 아이콘이 이 문제를 매우 형상적으로 설명합니다.

마치 요리할 때마다 냄비를 깨끗이 씻어서 찬장에 넣었다가, 다음에 요리할 때 다시 꺼내는 것처럼, 이런 왕복 과정은 너무 시간이 낭비됩니다!

Slide 42: 磁盘IO开销大

现在我们来看第二个问题：
磁盘IO开销大。
图中显示MapReduce需要将中间结果写入磁盘，
那个硬盘的图标很形象地说明了这个问题。

就像每次做菜都要把锅洗干净放回柜子(guìzi)，
下次做菜再拿出来一样，
这个来回的过程太浪费时间了！

Slide 43: 높은 지연의 2단계 문제

세 번째 문제는 지연이 높다는 것입니다.
여기서 MapReduce의 엄격한 2단계 실행 모드를 보여줍니다: 01 Map 단계와 02 Reduce 단계.
이런 계단식 설계는 간단명료하지만, 큰 지연을 초래합니다.
마치 공장의 조립 라인처럼, 이전 공정이 완료되지 않으면 다음 공정은 시작할 수 없어서 전체 효율이 매우 낮습니다.

Slide 44: MapReduce工作流程

우리는 MapReduce의 구체적인 작업 흐름을 봅니다.
작업은 먼저 Map 함수로 처리되고, 그 다음 중간 결과를 출력한 후, Reduce 함수로 처리됩니다.
이 흐름도는 매우 명확하지만, 문제점도 드러냅니다: 직렬화된 처리 방식과 빈번한 디스크 I/O.

Slide 45: 并行处理示意图

MapReduce가 병렬 처리를 지원하지만(Map1, Map2, Map3와 Reduce1, Reduce2, Reduce3는 병렬로 실행될 수 있음),
두 단계 사이는 반드시 직렬이어야 하는데, 이것이 바로 병목 현상입니다.

마치 많은 사람이 외나무 다리를 건너는 것처럼, 각 사람은 빠르지만 다리가 좁아서 전체 속도는 여전히 매우 느립니다.

Slide 43: 延迟高的两阶段问题

第三个问题是延迟高。
这里展示了MapReduce严格的两阶段执行模式：01 Map阶段和02 Reduce阶段。
这种阶梯式的设计虽然简单明了，
但是造成了很大的延迟。
就像工厂的流水线，如果前一个工序 (gōng xù)没完成，后面的工序就不能开始，
整体效率会很低

Slide 44: MapReduce工作流程

这里我们看到了MapReduce的具体工作流程。

任务首先经过Map函数处理，然后输出中间结果，再由Reduce函数处理。

这个流程图很清楚，但也暴露了它的问题：串行化(chuànxínghuà)的处理方式和频繁(pínfán)的磁盘(cípán) IO

Slide 45: 并行处理示意图

虽然MapReduce支持并行处理（Map1、Map2、Map3和Reduce1、Reduce2、Reduce3可以并行执行），

但是两个阶段之间必须串行，
这就是瓶颈(píng jǐng)所在。

就像一群人过独木桥(mùqiáo)，每个人都很快，但是桥很窄(zhǎi)，所以整体速度还是很慢。

Slide 46: 等待时间问题

이 그림은 대기 시간 문제를 생생하게 보여줍니다. 모든 Map 작업이 완료되어야 Reduce 작업이 시작될 수 있습니다.

중간에 돋보기가 있고, 그 안에 "모든 Map 완료"라고 쓰여 있고, 그 다음 "대기"입니다.
이런 대기는 마치 교통 체증처럼, 뒤의 차가 이미 준비되었어도 앞의 차가 모두 통과할 때까지 기다려야 합니다.

--

Slide 46: 等待时间问题

这张图生动地展示了等待时间的问题。
所有的Map任务必须完成后，
Reduce任务才能开始。

中间有一个放大镜，里面写着"完成全部Map"，
然后是"等待"。
这种等待就像交通堵塞一样，
即使后面的车已经准备好了，
也必须等前面的车全部通过。

Slide 47: 연결 I/O 오버헤드

여기서 "연결의 I/O 오버헤드가 크다"는 문제를 자세히 설명합니다.

Map 후의 결과는 디스크에 써야 하고, 그 다음 Reduce가 디스크에서 읽습니다.

이 쓰기와 읽기 과정은 마치 두 사람이 정보를 전달하는데 쪽지를 쓰는 방식을 통해야 하는 것처럼, 효율이 당연히 매우 낮습니다.

현대 컴퓨팅에서 디스크 I/O는 가장 큰 성능 병목 현상 중 하나입니다.

Slide 48: MapReduce 결함 총정리

이것은 매우 좋은 정리 페이지로, 원형 차트로 MapReduce의 세 가지 주요 결함을 보여줍니다:

1. 높은 지연
2. 디스크의 I/O 오버헤드가 큼
3. 연결의 I/O 오버헤드가 큼

이 세 가지 문제는 마치 세 개의 큰 산처럼 MapReduce에 압박을 주어 마땅한 성능을 발휘할 수 없게 합니다.

Slide 47: 衔接IO开销

这里详细说明了"衔接的IO开销大"问题。

Map后的结果需要写入磁盘，

然后Reduce再从磁盘读取。

这个写入和读取的过程就像两个人传递信息，

必须通过写纸条的方式，

效率自然很低。

现代计算中，磁盘IO是最大的性能瓶颈(píng jǐng)之一。

Slide 48: MapReduce缺陷总结

这是一个很好的总结页面，

用圆形图表展示了MapReduce的三大缺陷：

1. 延迟高、
2. 磁盘的IO开销大、
3. 衔接(xiánjiē)的IO开销大。

这三个问题就像三座大山，

压(yā)在MapReduce身上，

让它无法发挥出应有的性能。

Slide 49: Spark 개선 방안

드디어 Spark가 등장할 시간입니다!
이 그림은 Spark가 어떻게 Hadoop의 결함을 개선하는지 보여줍니다.
Hadoop 로고에서 Spark 로고로 향하는데, 마치 구시대에서 신시대로 나아가는 것 같습니다.

Spark는 Hadoop을 완전히 뒤엎으려는 것이 아니라, 그 장점을 계승하고 결함을 개선하려는 것입니다.

Slide 50: 从函数到操作符

여기서 우리는 Spark의 첫 번째 개선을 봅니다: 간단한 Map과 Reduce 함수에서 더 많은 연산자로 확장.
마치 덧셈과 뺄셈만 있던 것에서 곱셈, 나눗셈, 제곱근, 삼각 함수 등으로 확장하는 것처럼, 표현력이 크게 향상되었습니다.

Slide 51: 丰富的操作类型

Spark는 풍부한 연산 타입을 제공합니다: Map, filter, Reduce, join, groupBy 등등.

이 다채로운 별 모양 패턴은 Spark 연산의 다양성을 잘 보여주는데, 마치 만화경처럼 각종 다양한 패턴을 만들 수 있습니다.

Slide 49: Spark改进方案

终于到了Spark登场的时候！
这张图显示了Spark如何改进Hadoop的缺陷(quē xiàn)。
从Hadoop的标志(biāo zhì) 指向Spark的标志，
就像从旧时代迈向(mài xiàng)新时代一样。

Spark不是要完全推翻(tuī fān)Hadoop，
而是要继承(jìchéng)其优点，
改进其缺陷。

Slide 50: 从函数到操作符

这里我们看到了Spark的第一个改进：
从简单的Map和Reduce函数扩展到更多的操作符。
就像从只有加减法扩展到有乘除、开方、三角函数 (sānjiǎo hánshù)
等等，表达能力大大增强了。

Slide 51: 丰富的操作类型

Spark提供了丰富的操作类型，包括Map、filter、Reduce、join、groupBy等等。

这个彩色的星形图案很好地展示了Spark操作的多样性，就像一个万花筒(wàn huā tǒng)，能够创造出各种各样的图案。

Slide 52: 听写的三个名词不确定

요약하면, Spark는 더 많은 연산 타입을 제공하여 표현력을 크게 향상시켰습니다.

"연산 타입이 더 많다"는 것은 "표현력이 더 강하다"로 이어지는데, 이것은 인과 관계입니다.
마치 요리사에게 더 많은 조미료와 도구를 제공하면, 자연스럽게 더 풍부한 요리를 만들 수 있는 것과 같습니다.

Slide 52:听写的三个名词不确定

总结来说，Spark通过提供更多的操作类型，大大增强了表达能力。

"操作类型更多"指向"表达能力更强"，这是一个因果关系。

就像给厨师(chú shī)提供了更多的调料和工具，自然能做出更丰富的菜肴(cài yáo)。

Slide 53: 메모리 계산 개념

이제 Spark의 두 번째 중대한 개선을 봅시다: 메모리 계산.

그림은 계산 과정 중 중간 결과가 메모리에 직접 저장됨을 보여주며, 그 녹색 메모리 막대 아이콘이 매우 형상적입니다.

이것은 마치 자주 사용하는 도구를 손 닿는 곳에 두어, 필요할 때 바로 가져갈 수 있는 것처럼, 매번 창고에 가서 찾을 필요가 없습니다.

Slide 53: 内存计算概念

现在我们来看Spark的第二个重大改进：内存计算。

图中显示计算过程中的中间结果直接存储在内存中，
那个绿色的内存条图标很形象。

这就像把常用的工具放在手边，
需要时随手就能拿到，
而不用每次都去仓库里找。

Slide 54: 内存计算优势

메모리 계산의 우수성은, 중간 결과에 다시 접근해야 할 때 메모리에서 직접 읽을 수 있어 낮은 지연을 실현한다는 것입니다.

그림의 작은 로켓 🚀은 속도 향상을 상징합니다.

이것은 마치 자주 사용하는 전화번호를 머릿속에 기억해서, 필요할 때 즉시 떠올릴 수 있는 것처럼, 매번 전화번호부를 찾을 필요가 없습니다.

Slide 55: 迭代运算效率

메모리 계산은 특히 반복적인 반복이 필요한 알고리즘에 적합하며, 반복 연산 효율을 효과적으로 향상시킬 수 있습니다.
그림의 순환 화살표는 반복의 개념을 잘 보여줍니다.
마치 피아노를 연습하는 것처럼, 매번 피아노를 옮기고 조율해야 한다면 효율이 매우 낮을 것입니다.

만약 피아노가 그 자리에 있어서 언제든지 연습할 수 있다면, 효율은 자연스럽게 훨씬 높아집니다.

Slide 54: 内存计算优势

内存计算的优势在于，当需要再次访问中间结果时，可以直接从内存中读取，实现了低延迟。

图中的小火箭(huǒjiàn) 🚀 象征着速度的提升(tí sheng)。

这就像把常用的电话号码记在脑子里，需要时立刻就能想起来，而不用每次都翻电话本。

Slide 55: 迭代运算效率

内存计算特别适合需要反复迭代的算法，可以高效提高迭代运算效率。

图中的循环箭头很好地展示了迭代的概念。

就像练习钢琴，如果每次都要重新搬琴、调音，那效率会很低；

如果琴就在那里，随时可以练习，效率自然会高很多。

Slide 56: DAG 작업 스케줄링

Spark의 세 번째 중요한 특성은 DAG(유향 비순환 그래프) 기반의 작업 스케줄링 실행 메커니즘입니다.

오른쪽 그림은 DAG의 구조를 보여줍니다: A→B→C→E, B→D→C, F→G.
이것은 마치 상세한 시공 도면처럼, 작업자들에게 어떤 순서로 각 작업을 진행해야 하는지 알려주어 불필요한 대기와 반복을 피합니다.

Slide 57: MapReduce 반복의 문제

이제 구체적인 예를 통해 MapReduce가 반복 알고리즘을 처리할 때의 문제를 봅시다.

반복1이 결과1을 생성하고, 그 다음 디스크에 써야 합니다.

이 하드 디스크 아이콘은 매 반복마다 디스크 작업을 수행해야 한다는 것을 상기시켜 주는데, 이것이 효율이 낮은 근원입니다.

Slide 56: DAG任务调度

Spark的第三个重要特性是基于DAG（有向无环图）的任务调度执行机制。

右边的图形展示了DAG的结构：
A→B→C→E，B→D→C，F→G。
这就像一张详细的施工图，
告诉工人应该按什么顺序进行各项工作，
避免了不必要的等待和重复。

Slide 57: MapReduce迭代的问题

现在我们通过一个具体的例子来看MapReduce在处理迭代算法时的问题。

迭代1产生结果1，然后需要写入磁盘。
这个硬盘图标提醒我们，

每次迭代都要进行磁盘操作，这是效率低下的根源。

Slide 58: 多次迭代的累积问题

여러 번의 반복이 필요할 때(반복1, 반복2), 매번 디스크 읽기/쓰기 과정을 반복해야 합니다.

결과1, 결과2 모두 디스크에 저장되어야 하므로, 이렇게 누적되면 I/O 오버헤드가 매우 커집니다.

마치 이사하는 것처럼, 매일 한 번씩 이사해야 한다면 정말 죽도록 피곤할 것입니다!

Slide 59:迭代算法的重要性

여기서 반복적인 반복이 필요한 두 가지 전형적인 알고리즘을 나열합니다:
로지스틱 회귀와 담금질 기법 알고리즘.
이러한 알고리즘은 머신러닝과 최적화 문제에서 매우 중요하며, 처리 효율이 낮으면 전체 프로젝트의 진행이 영향을 받습니다.
문제는 여러분들은 아직 로지스틱회귀와 시뮬레이티드 어닐링 알고리즘의 동작 특성을 모르고 있는 것이 문제입니다.
여기서는 그냥 인공지능학습에서 훈련시에 반복작업이 많다는 정도만 알아두세요.

Slide 60: MapReduce实现效率问题

그림의 순환 화살표는 반복 과정을 보여주며, 알고리즘이 "수렴하지 않았을 때" 계속 반복해야 합니다.
MapReduce의 구현 방식은 "효율이 매우 낮은" 결과를 초래하는데, 이 빨간색 원은 특히 이 문제의 심각성을 강조합니다.

Slide 58: 多次迭代的累积问题

当需要进行多次迭代时（迭代1、迭代2），
每次都要重复磁盘读写的过程。

结果1、结果2都要存储在磁盘上，
这样累积(lěi jī)下来，IO开销就非常大了。

就像搬家，如果每天都要搬一次，
那真的会累死人的！

Slide 59:迭代算法的重要性

这里列举了两个典型的需要反复迭代的算法：
逻辑斯蒂回归和模拟退火算法。

这些算法在机器学习和优化问题中非常重要，如果处理效率低，
那整个项目的进度都会受到影响。
问题在于，你们还不了解逻辑回归和模拟退火算法的工作特性。
在这里，大家只需要知道人工智能在训练过程中需要大量重复操作就可以了。

Slide 60: MapReduce实现效率问题

图中的循环箭头显示了迭代过程，
当算法"没有收敛"时需要继续迭代。
MapReduce的实现方式导致"效率非常低"，
这个红色圆圈特别强调了这个问题的严重性。

Slide 61: MapReduce单次迭代详情

여기서 MapReduce 단일 반복의 과정을 자세히 보여줍니다: 반복1이 Map + Reduce를 통해 결과1을 생성하고, 그 다음 디스크에 씁니다.

이 과정은 매우 간단해 보이지만, 수백 수천 번의 반복이 필요할 때 문제가 드러납니다.

Slide 62: 多次迭代的重复开销

반복1과 반복2의 과정이 완전히 동일하며, 매번 같은 디스크 I/O 작업을 반복해야 합니다.
이런 반복은 마치 외출할 때마다 새로 옷을 입고 머리를 빗어야 하는 것처럼, 매번의 시간은 길지 않지만 누적되면 상당합니다.

Slide 61: MapReduce单次迭代详情

这里详细展示了MapReduce单次迭代的过程：迭代1通过Map + Reduce产生结果1，然后写入磁盘。

这个过程看起来很简单，
但是当需要成百上千次迭代时，
问题就显现出来了。

Slide 62: 多次迭代的重复开销

迭代1和迭代2的过程完全相同，
每次都要重复同样的磁盘IO操作。

这种重复就像每次出门都要重新穿衣服、梳头发(shū tóu fa)一样，虽然每次时间不长，
但累积(lěi jī)起来就很可观了。

Slide 63: Spark의 해결 방안

Spark는 "데이터 낙지 회피"와 "디스크에 쓰지 않음"으로 이 문제를 해결합니다.

그림의 금지 표시는 Spark의 태도를 매우 명확히 보여줍니다: 불필요한 디스크 작업을 거부합니다!

이것은 마치 매일 이사하지 않기로 결정하고, 고정된 장소를 찾아 정착하는 것과 같습니다.

Slide 63: Spark的解决方案

Spark通过"避免数据落地"和"不写磁盘"来解决这个问题。

图中的禁止标志很清楚地表明了Spark的态度：拒绝(jù jué)不必要的磁盘操作！

这就像决定不再每天搬家，
而是找一个固定的地方住下来。

Slide 64: 流水线优化概念

Spark는 또한 파이프라인 최적화를 구현하여, 데이터 생성과 데이터 사용이 더 잘 협력할 수 있게 합니다.
연산1이 데이터를 생성하면, 연산2가 즉시 데이터를 사용하고, 중간의 데이터 교환이 메모리에서 진행됩니다.

이것은 마치 공장의 현대화된 조립 라인처럼, 각 환절이 긴밀히 협력하여 불필요한 중단이 없습니다.

Slide 64: 流水线优化概念

Spark还实现了流水线优化，
使得数据生成和数据使用可以更好地配合。
操作1生成数据，
操作2立即使用数据，
中间的数据交换在内存中进行。

这就像工厂的现代化流水线，
各个环节紧密(jǐnmì)配合(pèihé) ，
没有不必要的停顿(tíngdùn) 。

Slide 65: 执行流程对比详图

이 몇 개의 그림은 Hadoop MapReduce와 Spark의 실행 흐름을 자세히 비교합니다.

Hadoop은 반복적인 HDFS 읽기와 쓰기가 필요하지만, Spark는 중간 결과를 "메모리에 저장"할 수 있어 I/O 작업을 크게 줄입니다.

이런 비교는 마치 고대의 역참 전달과 현대의 전화 통신을 비교하는 것처럼, 효율이 완전히 다른 차원에 있습니다.

Slide 66: 执行流程对比详图

Slide 65: 执行流程对比详图

这几张图详细对比了Hadoop MapReduce和Spark的执行流程。

Hadoop需要反复的HDFS读取和写入，而Spark可以将中间结果"存储在内存中"，大大减少了IO操作。

这种对比就像比较古代的驿站(yì zhàn)传信和现代的电话通信一样，
效率完全不在一个层次上

Slide 66: 执行流程对比详图

这几张图详细对比了Hadoop MapReduce和Spark的执行流程。
Hadoop需要反复的HDFS读取和写入，而Spark可以将中间结果"存储在内存中"，大大减少了IO操作。
这种对比就像比较古代的驿站传信和现代的电话通信一样，
效率完全不在一个层次上

Slide 67: 执行流程对比详图

这几张图详细对比了Hadoop MapReduce和Spark的执行流程。
Hadoop需要反复的HDFS读取和写入，而Spark可以将中间结果"存储在内存中"，大大减少了IO操作。
这种对比就像比较古代的驿站传信和现代的电话通信一样，
效率完全不在一个层次上。

Slide 68: 성능 비교 실험 결과

마지막으로 우리는 놀라운 실험 결과를 봅니다:
로지스틱 회귀 알고리즘을 실행할 때, Hadoop은 110초가 필요하지만 Spark는 단지 0.9초만 필요합니다!

이 엄청난 성능 차이는 막대 그래프로 명확히 보여집니다.

Spark는 작은 왕관 👑도 쓰고 있어서, 그것이 성능의 왕임을 나타냅니다!

이것은 마치 걷기와 비행기 타기의 속도 차이를 비교하는 것처럼, 완전히 다른 차원에 있습니다.

Slide 68: 性能对比实验结果

最后，我们看到了令人震惊(lìng rén zhèn jìng)的实验结果：
执行逻辑斯蒂回归算法，Hadoop需要110秒，而Spark只需要0.9秒！

这个巨大的性能差异用柱状图清楚地展示出来。

Spark还戴着一个小王冠(wáng guān) 👑，说明它是性能之王！

这就像比较步行和坐飞机的速度差异一样，完全不在一个层次(céngcì)上。

퀴즈 정답

Slide 69: ✅ 정답: b

Slide 70: ✅ 정답: C

Slide 71: ✅ 정답: d

Slide 72: ✅ 정답: D

Slide 73: ✅ 정답: C

Slide 74: 정답: B) 메모리 계산

Slide 75: 정답: C) 유향 비순환 그래프

Slide 76: 정답: C) MLlib

Slide 77: 정답: B) 아니요, Spark는 주로 MapReduce를 대체하며, 여전히 HDFS가 필요합니다

Slide 78: ✅ 정답: D

Slide 79: 수업 마무리

좋습니다, 이번 수업은 여기까지입니다!
오늘 우리는 "빅데이터 시대"의 연원을 알아보았습니다. 정보화 물결, 세 가지 지원 기술부터 데이터가 어떻게 생성되고 어떻게 진화하는지까지 배웠습니다.
다음 수업에서는 "빅데이터의 핵심 기술"에 대해 계속 탐구하겠습니다.
마지막 질문 하나: 오늘 여러분은 사실 매일 "데이터를 생산"하고 있다는 것을 발견하셨나요? 😉

감사합니다 여러분, 수업 끝! 👏👏👏

Slide 69:✅ 正确答案：b

Slide 70:✅ 正确答案：C

Slide 71:✅ 正确答案：d

Slide 72:✅ 正确答案：D

Slide 73:✅ 正确答案：C

Slide 74:正确答案：B) 内存计算

Slide 75:正确答案：C) 有向无环图

Slide 76:正确答案：C) MLlib

Slide 77:正确答案：B) 不会，Spark主要取代MapReduce，仍需要HDFS

Slide 78:✅ 正确答案：D

Slide 79:
好了，我们本节课到这里就结束啦！
今天我们认识了"大数据时代"的来龙去脉，从信息化浪潮、三大支撑技术，到数据如何产生、如何演变。

下节课我们会继续探讨"大数据的关键技术"。
最后提问一下：你今天有没有发现，自己每天其实也在"生产数据"呢？😉

谢谢大家，下课！👏👏👏